THE UNIVERSITY *of* EDINBURGH

MOTION PLANNING FOR SELF-RECONFIGURING ROBOTIC SYSTEMS

Abstract. Robots that can actively change morphology offer many advantages over fixed shape, or monolithic, robots: flexibility, increased maneuverability and modularity. So called self-reconfiguring systems (SRS) are endowed with a shape changing ability enabled by an active connection mechanism. This mechanism allows a mechanical link to be engaged or disengaged between two neighboring robotic subunits. Through utilization of embedded joints to change the geometry plus the connection mechanism to change the topology of the kinematics, a collection of robotic subunits can drastically alter the overall kinematics. Thus, an SRS is a large robot comprised of many small cooperating robots that is able to change its morphology on demand. By design, such a system has many and variable degrees of freedom (DOF).

To gain the benefits of self-reconfiguration, the process of morphological change needs to be controlled in response to the environment. This is a motion planning problem in a high dimensional configuration space. This problem is complex because each subunit only has a few internal DOFs, and each subunit's range of motion depends on the state of its connected neighbors. Together with the high dimensionality, the problem may initially appear to be intractable, because as the number of subunits grow, the state space expands combinatorially. However, there is hope. If individual robotic subunits are identical, then there will exist some form of regularity in the resulting state space of the conglomerate. If this regularity can be exploited, then there may exist tractable motion planning algorithms for self-reconfiguring system.

Existing approaches in the literature have been successful in developing algorithms for specific SRSs. However, it is not possible to transfer one motion planning algorithm onto another system. SRSs share a similar form of regularity, so one might hope for mathematical tools that would identify the common properties that are exploitable for motion planning. So, while there exists a number of algorithms for certain subsets of possible SRS instantiations, there is no general motion planning methodology applicable to all SRSs.

In this thesis, we first evaluate the best existing general motion planning techniques applied to SRS motion planning problem. Greedy search, simulated annealing, rapidly exploring random trees and probabilistic roadmap planning were found not to scale well with respect to computation time as the number of subunits in the SRS increased. The planners performance was limited by the availability of a good general purpose heuristic. There does not currently exist a heuristic which can accurately guide a path through the search space toward a far away goal configuration.

Second, we show that computationally efficient reconfiguration algorithms do exist by development of an efficient motion planning algorithm for an exemplary SRS, the Claytronics formulation of the Hexagonal Metamorphic Robot (HMR). The developed algorithm was able to solve a randomly generated shape-to-shape planning task for the SRS in near linear time as the number of units in the configuration grew. Configurations containing 20,000 units were solvable in under ten seconds on modest computational hardware. The key to the success of the approach was discovering a subspace of the motion planning space that corresponded with configurations with high mobility. Plans could be discovered in this sub-space much more readily because the risk of the search entering a blind alley was greatly reduced.

Thirdly, in order to extract general conclusions, we analyzed the computationally efficient subspace discovered in the HMR state space, and other efficient subspaces utilized in other works, using graph theoretic methods. We find that the high mobility is observable as an increase in the state space's Cheeger constant, which was estimated using a local sampling procedure. Furthermore, state spaces associated with an efficient motion planning algorithm are well ordered by the graph minor relation. This opens the door for automated methods for discovery of efficient planning subspaces for SRS motion planning, and suggest an avenue for future work toward development of a general purpose SRS motion planner compiler.

Contents

1. Introduction 5
2. Self-Reconfiguring Robot Architectures 8
3. Previous Work On Discrete SRS Planning 11
3.1. Compressible 12
3.2. Lattice 15
3.3. Chain Type 19
3.4. Hybrid 20
3.5. General Theory 21
4. General Motion Planning Methods for Self-Reconfiguration Planning [44] 24
4.1. Models 25
4.2. Heuristics 26
4.3. Planners 30
4.4. Experiments 33
4.5. Discussion 37
4.6. Conclusion 39
5. An Efficient Algorithm for Self-Reconfiguration Planning in a Modular Robot [45] 40
5.1. Surface model 40
5.2. Planning 43
5.3. Results 53
5.4. Discussion 54
6. A Characterization of the Reconfiguration Space of Self-Reconfiguring Robotic Systems [46] 55
6.1. Introduction 55
6.2. Preliminaries 55
6.3. Background 57
6.4. A Surface-to-Surface Planner 58
6.5. The Surface Space is Highly Connected 62
6.6. Graph Minor Sub-Structure 66
6.7. Discussion 77
6.8. Conclusion 79
7. Conclusion 80
8. Future Work 83
Appendix 1: Random configuration generation 92
1.1. Generator 1 94
1.2. Generator 2 95
1.3. Candidate Generator 3 95
1.4. Discussion 96
Appendix 2: The C4.5 data mining algorithm and its application 99

2.5. The ID3 algorithm 99
2.6. Application 100
2.7. Results 101
2.8. Conclusion 102
References 106

1. Introduction

Self-reconfiguring robots are robotic systems that are comprised of many cooperating robotic subunits. Thus, a self-reconfiguring robot is specialized form of a modular robot. The unique feature of self-reconfiguring robots that distinguish them from other modular robotic systems is the ability of subunits to actively engage/disengage a mechanical connection between each other. This ability allows the complete system to alter its physical topology and its kinematics. By chaining sequences of kinematic alterations together, a self-reconfiguring robotic system is able to drastically alter its overall morphology.

A robot that can autonomously alter its morphology has many advantageous over an alternative monolithic robotic system [53, 82]. For example, the robot can alter shape to suit environmental challenges *e.g.* turning into a snake-like configuration in order to squeeze through a narrow passage, or dynamically altering the workspace to manipulate an awkward work piece. The ability of self-reconfiguring robots to move constituent components around the internal structure could be utilized to replace broken parts without human intervention; broken parts can be reconfigured off the structure and replaced with working versions. In addition, accuracy [43] and strength [81] can be autonomously scaled through the use of parallel actuation.

There are, however, several technical obstacles that prevent self-reconfiguration technology from being adopted. One important set of issues regard the engineering of SRSs. SRSs inherently contain a lot of redundancy in the hardware, and this makes the power-to-weight ratio a problem for realization. The repetition of functionality also makes the robotic units cost more, so SRSs development is currently very expensive. These are hard issues being addressed by experts in mechanical engineering, materials science etc. In addition to these issues there is a unique planning in for the domain.

The issue addressed here is the question of how to control the robotic conglomerate. For an SRS to be a beneficial alternative to a monolithic robot implementation, it must use its unique ability to achieve something that the monolithic robot could not. For an SRS this means reconfiguring in response to some form of input. So far there is no agreement within the academic community on what form the input should be. Most researchers to date have focused on the problem of given a desired configuration, the SRS must be able to autonomously change from its current configuration into the desired one [2, 72, 11, 66, 5]. This is the main focus of the thesis here, but there are other possibilities, such as: move a held tool while respecting some workspace (but ignoring the exact placement of the body units), move a tool at a specific level of strength/accuracy [81, 43] or other high level commands such as locomote the entire aggregate over rough terrain [7].

This thesis concentrates on the shape-to-shape reconfiguration tasks. That is, determining a sequence of individual subunit commands that change the whole from a starting shape into a desired goal shape. A general purpose shape-to-shape motion planner could be used as a low level building block for higher level control systems for SRSs, in the same way that the PID controller if often the last computation in building a navigation system.

Control of continuous system is often driven by an error term which can be differentiated or integrated. SRS are characterized by their ability to connect/disconnect components from one another; connectivity changes are discrete events in time, and so SRSs are combinatorial objects. Because the systems dynamics are driven by discrete events, the continuous concepts used in classical control have no direct analogy. *For an SRS, in order to reduce the difference between a current shape and a desired shape, there no simple derivative to take to provide a direction to move.* As we will see later, it turns out that determining a move to apply to drive an SRS towards a goal is, in-fact, a very challenging problem indeed, with no general computationally efficient solution available.

Generally speaking, combinatorial objects are harder to plan with. If all combinations need to be enumerated, then the computation time to explore the space scales exponentially with problem complexity. However, SRS are comprised of repetitions of identical unit classes, and intuition suggests there must be some form of exploitable structure in the problem space to take advantage of.

In chapter 2 we describe the broad categories of SRS architectures. We describe how the principles of motion planning for lattice based SRS is important for a diverse range of SRS architectures, including those that might be useful in an engineering context. This thesis is focused on finding general principles that apply to motion planning for all lattice based SRS.

In chapter 3 we review salient planning research for lattice based SRSs. A number of works have achieved $O(n^2)$ worst case time complexity for a variety of different SRS. However, it is not clear from the literature that modern sampling based planning methodologies might not solve the general problem.

In chapter 4 we evaluate a wide range of general planning methodologies, including rapidly-exploring random trees and probabilistic roadmap planning, to two different lattice SRS models derived from the hexagonal metamorphic robot (HMR) [13], in the hope that new advances in planning might prove useful in the SRS planning domain. However, this investigation shows that general search, even modern methodologies, are not feasible for solving shape-to-shape reconfiguration tasks. There is no room for error, and therefore backtracking, when the problem scales above 30 DOFs.

In chapter 5 we fill a gap in the literature by developing a very efficient planner for the HMR. We show this planner empirically has close to $O(n)$ average case performance (clearly sub quadratic). This planner, unlike many other efficient algorithms, does not meta-modularize the state space into coarse planning units, and thus can solve a high percentage of the underlying state space offered by the particular HMR motion constraints. The key reason for its efficiency was the identification of a subspace within the underlying state space that was large and highly amenable to efficient planning.

While both chapter 5 and 4 focus on specific SRS models. In chapter 6 we abstract the problem using graph theoretic tools. The efficient state space of chapter 6 is analyzed as an undirected, unlabeled graph. By ignoring the labeling of vertices of the state space, the SRS architecture specific information is lost, and general conclusions, applicable to a wide range of SRSs, are appreciable. It was discovered that state spaces relating to existing efficient algorithms, have a low Cheeger

constant, suggesting the state spaces tend not to contain bottlenecks that could lead to large local minima. Furthermore, efficient state spaces have an special ordering across the family of spaces generated sequentially by adding a subunit, which is not apparent in state spaces with no known efficient algorithms. An efficient state space, relating to the motion of n subunits, is a graph minor of the state space corresponding to the motion of $n+1$ subunit. The consequence of this ordering is that high dimensional planning problems can be decomposed and solved in a computationally efficient manner. In chapter 8 we sketch how future researchers might utilize the graph minor ordering observation to build a motion planning compiler for general SRS motion models.

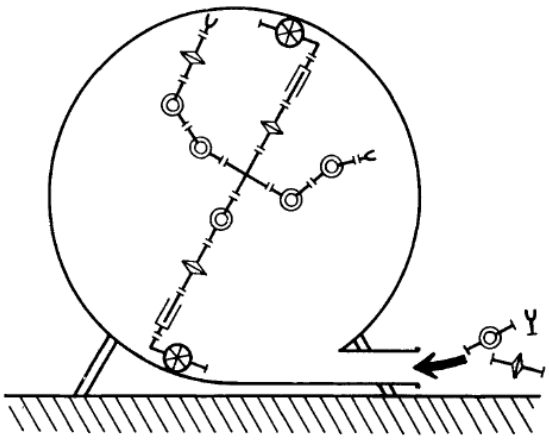


FIGURE 2.1. An example of what was hoped for CEBOT, taken from[24]. In the vision, individual modules could be inserted into a pipe opening, which would then self-assemble into a complex pipe cleaning robot. A variety of modules were sketched, wheel modules, articulated modules and gripper modules.

2. Self-Reconfiguring Robot Architectures

Modularity and flexibility have been enshrined principles in engineering since the industrial revolution. Even quite early in robot technology it was understood that the same robot manipulator could do multiple tasks if it had the ability to change tools [37]. Tool changers allowed robotic arms and tools to be interchangeable, and they were the exemplars of modularity within robotic technology. In 1988 Fukuda and Nakagawa extended the idea of swapping tools on a fixed manipulation platform, to swapping the joints too [23]. This idea was implemented in CEBOT, the first attempt at producing a self-reconfiguring robot [24].

CEBOT was comprised of many different types of modules each with their own specialization, and an analogy was drawn to cells within multi-cellular organisms (hence the name CEllular roBOT). Every major type of joint had its own unique module, and their were module types to provide locomotion (figure 2). While this approach provides the ultimate flexibility in what can be built from the constituent components, determining what is a good configuration to carry out a specific task requires picking a configuration from a huge state space, which was computationally infeasible.

Two following robotic architectures, Polypod [81] and the metamorphic robot [12] were shortly formulated afterwards and both approaches similarly differed from the heterogeneous CEBOT by being either unipartite (Metamorphic) or bipartite (Polypod). By using only one or two classes of modules, the number of possible configurations was drastically reduced. The two approaches differed, however, in important ways too. Chirikjian's metamorphic robot defined, what is now referred to as, a lattice based SRS, while Yim's Polypod became a prototypical example of a chain type SRS.

Chain type SRS are characterized by the structure containing sub-sequences of units that kinematically resemble robotic arms, in that they contain continuous joints. Sometimes two classes of

units are used in chain type SRSs, one class contains joints and the other class is unactuated, but has more than two connectors. The unactuated component can connect to three or more neighbors and so introduces branching into the connectivity topology. Yim's Polypod was this type of chain SRS, with the unactivated component being a cube.

Chain type self-reconfiguring robots are able to reach and interact with objects in 6DOF real space, and thus could potentially carry out the same type of tasks industrial robot arms are currently used for. Lattice type self-reconfiguring robots, in contrast, can only be located at locations on some regular lattice. Lattice based robots lose mobility, but have the benefit of a simpler state space. Yet despite the model of reconfiguration being the simplest, lattice SRS are still difficult to plan for. Lattice based SRSs, while probably not being of direct practical in engineering, provide a mathematical benchmark that must be vaulted before tackling the planning problem for more pragmatic SRSs.

Murata *et al.* developed two more lattice based SRSs (Fracta and Fracta3D) [40, 54] before a third category of SRS was developed by D. L. Rus, M. Vona in the form of an SRS called Crystalline [66]. Crystalline subunits featured linear actuators which altered the two side lengths of the rectangular subunits. Crystalline was known as a unit-compressible system, and was one of the earliest systems that efficient, scalable planning algorithms were developed for [65]. Several other unit compressible SRS have since been developed [74].

Unit compressible systems, in general, are aligned with an underlying lattice for planning. This makes them strongly related to lattice based systems. The important difference, is that the volume of each subunit can be actively adjusted whereas in other lattice based systems each units size is completely fixed. The packing then of a unit-compressible system on the embedding lattice has to accommodate different sized subunits placed on a fixed sized lattice.

An implication of laying units upon a lattice (including the unit-compressible case) is subunits have their position and orientation discretized. Chain SRSs do not have this restrictions, and are able to assume standard manipulator morphologies that are like traditional robot arms [31, 10]. This is a significant advantage for engineering applications that chain type SRSs posses. It became apparent though, that the increased flexibility of chain SRS caused problems when subunits tried to position themselves to engage the connection mechanism [34, 60, 59, 64, 71]. Connection mechanism reliability was reduced for chain SRSs. This issue inspired a new type of SRS called the hybrid architecture.

Hybrid systems are chain SRSs that restrict themselves to a lattice when reconfiguring. These systems often butt against one-another physically when in a lattice adhering shape, and thus do not require the actuators to hold position. This passive positioning removes a large source of positional error found in chain SRSs caused by actuator control in-precision. Hybrid systems [55, 68] have so far been constructed with rotational DOFs that allow the system to manipulate using continuous variables. The body of the subunits though, are designed around a tight natural packing of the units on a lattice. The idea is that subunits alter their morphology when aligned with the lattice, but after the desired connection topology is achieved, the system can leave the lattice restriction and operate in more general space. Engaging the connection mechanism is more reliable, because subunits are

butted against one another, and positional errors caused by gravity are zeroed passively. This results in systems that can change between a 4 legged walker and a long snake and other radical locomotive phase transition [42], feats that has not been possible on chain type SRSs so far (and impossible for lattice SRSs).

Hybrid systems appear to be the best physical instanciation of an SRS which has the potential to be useful in the field. During changes of morphology, hybrid SRSs, for planning purposes, adhere to a lattice. However, the planning problem for hybrid systems is complex compared to the basic lattice SRSs first developed. Early lattice systems could be represented as a labeled lattice. Only two labels were required, occupied or empty, and it was assumed that subunits could actively dock with any adjacent subunit. By contrast, successful hybrid systems such as M-TRAN, each subunit occupies two lattice locations simultaneously, and have different types of faces that can intact with adjacent subunits in different ways. Furthermore, these faces can be actively orientated in different relative poses. While both lattice and hybrid SRSs can be represented by labels on a lattice and motion represented by labeling operations, the number of labels, and the number of possible relabeling operations are greatly increased for hybrid systems built so far. This increase in complexity of the motion rules, has meant that human algorithm designers have only been successful in creating efficient, scalable motion planners for the simpler types of lattice based systems.

It is desirable that we develop general purpose motion planning algorithms, so that we can build motion plan synthesizers for complex SRSs like M-TRAN. These are the forms of system that are most likely to be deployed in the field and perform useful tasks. The core shape-to-shape reconfiguration problem for hybrid SRSs shares similarities to the planing task for basic lattice based SRSs. By understanding lattice based systems better, we hopefully will reveal new insights that will have practical ramifications in the hybrid SRS planning use-case.

3. Previous Work On Discrete SRS Planning

Development of motion planning algorithms has often been carried in parallel to development of a physical SRS robot prototype. Thus, the domain of the planner tends to be focused on the particular set of local constraints that the developing robot has. This has tended to segment motion planning algorithms according to the type of the SRS architecture used, be it unit-compressible, lattice, chain or hybrid. Unit-compressible and lattice systems are represented as discrete combinatoric objects (e.g. a labeled lattice).

Chain and hybrid systems contain rotational joints. The movement of a single joint can alter the relative arrangement of all subunits either side of the joint. This global transform can mean the connectivity graph can alter drastically per individual reconfiguration event. Optimal relabeling in this scenario has been shown to be NP-hard [30], and may mean planning is a different problem than in the lattice case. However, chain and hybrid SRS systems can artificially restrict their state space to a lattice, so insights into motion planning for lattice based SRS are relevant for all the architectures so far mentioned.

When SRS are viewed as combinatoric objects, it is natural to use motion planning techniques from classic AI, such as search. The spatial aspect of SRS also permits the use of graph theoretic metrics (spanning tree, Hamiltonian paths etc.) derived from the placement of the labels on the lattice e.g. the connectivity graph of occupied locations. Much of SRS planning literature can be broadly summarized as finding the right set of graph metrics to derive a heuristic that makes a search run fast.

The capability of the subunits in the SRS though, strongly affect the state space and therefore what the heuristic should be composed of. Some SRS superficially appear have the same gross behavior, such as subunits moving only over the perimeter, but subtle differences, such as certain local scenarios that prevent a subunit from locomoting. Thus, the old adage "the Devil is in the detail" is particularly astute for development of SRS motion planners.

There are several different questions to ask when evaluating a planning algorithm.

- What are the time and memory requirements when forming a motion plan?
- Is the planner complete *i.e.* is it guaranteed to be capable of forming a plan between any start and goal configuration? Has the SRS state space been artificially restricted, in order to be complete?
- Is it parallel? Do subunits move one at a time or can multiple subunits move in each iteration of time?
- Does the planner utilize distributed computation, and in particular, is the distributed computation model designed to be run on the same SRSs topology of computation resources?

Early research focused on time complexity, but at the cost of restricting the state space severely. Later works either lessened state space restrictions, focused on distributing the computation to make further computational gains, or parallized the movements of subunits. While most research has explicitly used graph theoretic constructs, sometimes the inspiration of a planner has come

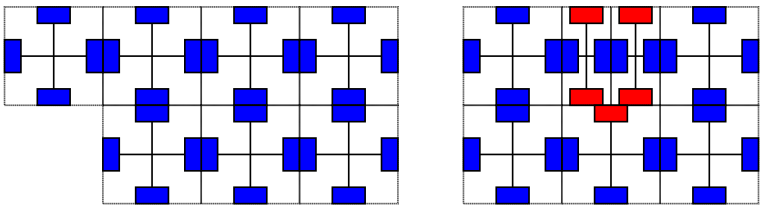


FIGURE 3.1. Crystalline model

from biology or physics, such as Shen's hormone inspired control, or Claytronics' movements of space by Brownian motion (novel for its stochasticity).

In this thesis our focus is on deterministic lattice based SRS motion planning, but there are other highly interesting architectures not yet discussed. White *et al.* suggest that a stochastic model of reconfiguration may be useful for designing an SRSs made of chemical components [79]. Flexible modular robots have been made from compliant joints [85]. SRSs have been built from holonomic vehicles, both wheeled [33, 52] and flying [57]. The field of swarm robotics deals with organizing large numbers of robots in a spatial domain, however, the difficult local constraints typical of SRSs are not present, so swarm robotics research focuses on different problems than those addressed here [51].

In the following subsections, salient motion planning algorithms are described that are of relevance to lattice based planning. While most planning research has focused on specific algorithms for specific SRSs, there have been a few attempts to find general insights across the domain of SRSs. Of particular note is Ghrist and Peterson's work, who developed the nomenclature we use extensively in this thesis. General insights are discussed in the final subsection.

3.1. **Compressible.** Unit compressible SRSs have subunits that can alter their linear diameters actively, commonly by a factor of 0.5. Despite the concept of unit-compressible modules being developed after Chain type and lattice based SRSs, efficient unit compressible planning algorithms ($O(n^2)$ or less) were available for this architecture first [66]. Furthermore, motion planning algorithms for unit-compressible SRSs have tended to lead the SRS field in general ever since. This suggests that unit-compressible systems are somehow easier to plan with; this could be for a fundamental mathematical reason, or it may be that unit-compressible modules are easier to mentally visualize in researchers minds. We are of the opinion that it is the latter.

Rus and Vona developed the first shape-to-shape motion planning algorithm with provable $O(n^2)$ bounds for the Crystalline unit-compressible SRS [66]. In the work a Crystalline module is presented as a 2D unit with rectangular unit with connectors on the edges. A subunit could halve its length in one of the two possible axes at any point in time. The set of units must remain connected at all times, see Figure 3.1. Their Melt-Grow algorithm features two important concepts, meta-modularization and reconfiguring via an intermediate configuration.

Reconfiguration planning was simplified by dividing the start and end shapes into 4x4 axis aligned grids of units (Figure 3.2), a strategy later given the term meta-modularization. This allowed reconfiguration planning to be split into two levels of abstraction, the motion of the n coarse 4x4

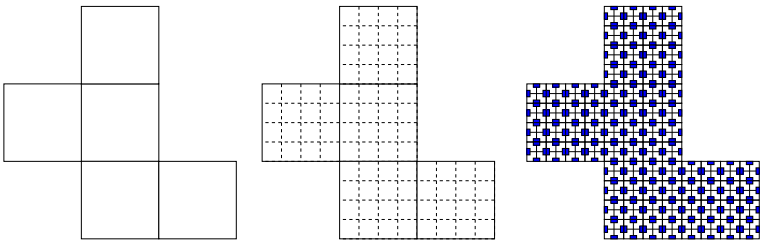


FIGURE 3.2. A, a shape. B the 4x4 axis aligned division of the shape. C the realization of the shape in Crystalline robots

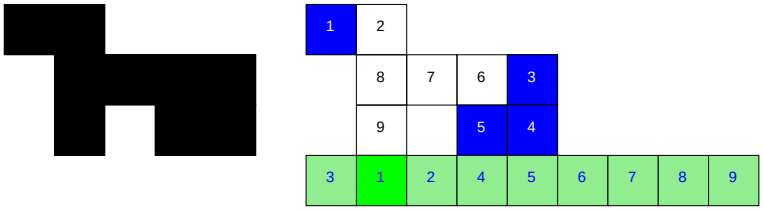


FIGURE 3.3. Melting the start configuration (black) into the intermediate location, green. Mobile meta-modules of the initial configuration are shown in blue (a removal of a blue subunit will not disconnect the structure in iteration 1). An example of a possible order of mobile unit migration is shown by numbering the removal locations in white and black, and the add locations in blue. The values of the removal locations were determined by performing a depth first search, starting at position 9, and counting down. When the lower left subunit (9) is chosen as the melt location, the 8th unit to move will always be the penultimate unit to move, because its removal earlier would disconnect the structure. Earlier subunits in the meting process as shown here (e.g. 1 and 2) may be melted in a slightly different order depending on implementation. However, 7 will always be melted after 6,2,3 or 4 because of the configurations structure.

groups of meta-modules to realize the reconfiguration task, and the movements of individual $16n$ units to realize the meta-module movements.

In order to change a start configuration S into a goal configuration G made out of 4x4 axis aligned grids of units, S was first "melted" into an intermediate configuration I and then grown into the goal configuration G, hence the name of the algorithm, melt-grow. Meta-modules were considered "mobile" if their removal did not disconnect the structure. The intermediate configuration was chosen to be a linear chain of meta-modules starting below the lowest meta-module of the start configuration. A subroutine was developed that allowed mobile units to move to any adjacent location in the current configuration (we shall describe this sub-routine later). It is easy to see that with the underlying sub-routine providing a high degree of mobility for meta-modules, it is simple to turn a given configuration into a linear chain (Figure 3.3) in $O(n)$. It is clear that identifying the location to start I can be done in $O(n)$ time by iterating all meta-module locations and keeping the lowest. Identifying the order in modules should be removed can be achieved by running a depth

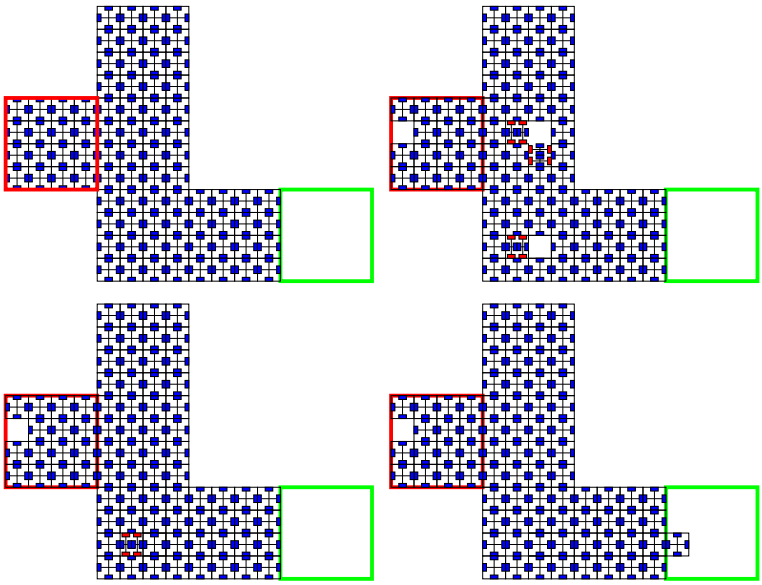


FIGURE 3.4. How an individual subunit of a meta-module can be transported through a path within the volume of a configuration. The meta-module highlighted in red is to be moved to the area highlighted in green. Top right, stage one: space is made at path turns by halving the length of two adjacent modules, as well as sucking one subunit from the remove location into the internal structure. Bottom left, different pair of units expand again into different location, in effect transporting the hole through the structure. Bottom right, the final pair of units expand expand, and a subunit appears in the target location.

first search starting at I and reversing the order in which subunits are encountered, at a one time cost of $O(n)$. Turning I into G is a similar algorithm but in reverse, also at a cost of $O(n)$. Overall the process of meting and growing requires an $O(n)$ number of underlying meta-module repositions.

The high level melt-grow algorithm delegates instructions to move meta-modules through paths within the structure. One unit could be removed from a location by absorbing its volume into the structure. By chaining these absorption moves together the volume of the unit could effectively be moved through the structure on via the a provided path. Note though, that the actual unit that appears in the target location is not the actual unit absorbed. Figure 3.4 shows an equivalent example provided by the authors in [66]. The middle pair of contracted units shown in the top right location cannot expand until the first does, and therefore the time to implement the complete move is $O(t)$ where t represents the number of turns in the path. It should be clear however, that some set of motions can be constructed that moves each of the 16 subunits along the internal path by moving them one at a time. The reason for the 4x4 meta-modularization becomes apparent, with this number of subunits in each 4x4 square, there is always enough maneuvering room for the center 2x2 subunits to butt against in order to propagate mass in any direction. The time required to reposition a meta-module is related to the number of turns (figure [66]), but as this quantity is bounded from above by the number of modules, n, the worst case execution time is $O(n)$. As

the higher level melt-grow algorithm requires $O(n)$ of these meta-module repositions, the overall algorithm has a worse case time complexity of $O(n^2)$.

Butler et al. developed the PacMan algorithm for the Crystalline robot. Their algorithm did not meta-modularize the space, nor planned to an intermediate configuration, at the cost that the algorithm sometimes became stuck. However, they had quite an intricate coordinating mechanism that allowed a number of subunits to be in motion at the same time. subunit also determined path through the structure to follow through network communication, so while the bounds for the algorithm were undetermined, the work was the first distributed, parallel algorithm for an SRS, and was successfully run on a hardware platform.

The Telecubes SRS was a similar SRS to the Crystalline robot but in 3D. In a series of works, Vassilvitskiil *et al.* [75] used 2x2x2 meta-modules to create a distributed algorithm where units moved in parallel. Unlike Butler *et al.* [5] this algorithm was complete, in the sense it would never get stuck, and unlike melt-grow it did not require an intermediate configuration to plan between. The algorithm had a provable worse case complexity of $O(n^2)$, but the authors noted this could not take into account the parallel actuation very well, and was likely to be a loose upper bound. The general approach to planning was very similar to the two previously mentioned methods, a set of motion primitives were developed that allowed meta-modules to virtually propagate through the structure. Their meta-module definitions differed by subunits being partially contracted by default, whereas in the Melt-grow algorithm and the PacMan algorithm the subunits were fully expanded. This subtle difference seemed to simplify the underlying motion primitives considerably, and lead to compact meta-module and less issues in the global planner (no need for an intermediate configuration to plan between).

3.2. **Lattice.** Lattice based SRS differ from unit-compressible by the subunits having a fixed volume. The location of each subunit must lie on unique locations on a discrete lattice. Thus the presence of a unit blocks the movement of another unit. Planning for lattice SRSs seems harder then, because the process of getting units out of the way to move another subunit on a path through the embedding space seems to require global coordination, as getting the units out of the way also may need other units to be got out of the way *ad infinitum*. Unit-compressible subunits in contrast have the ability to create space locally without disrupting the whole. That said, it is possible to construct a meta-module definition of a collection of lattice subunits that include space, and then run similar planning algorithms as were previously developed for unit-compressible SRSs.

Lattice based SRSs were invented by Chirikjian as a concept known as a Metamorphic robot [12]. For an SRS to be a metamorphic robot it had to be comprised of identical subunits, tightly packed on a lattice, remaining connected at all times. Although several different SRSs could satisfy this definition, their work on planning concentrated on the hexagonal metamorphic robot, named so because the embedding lattice the subunits reside on is hexagonal [13]. Their physical implementation of the robot was comprised of 6 bar linkages, which meant that a unit was able to move to an adjacent location if it had a robot neighbor to pivot upon, and that the moving unit was not an articulation vertex w.r.t the connectivity graph. Figure 3.5-A shows (top) the 6 bar linkage

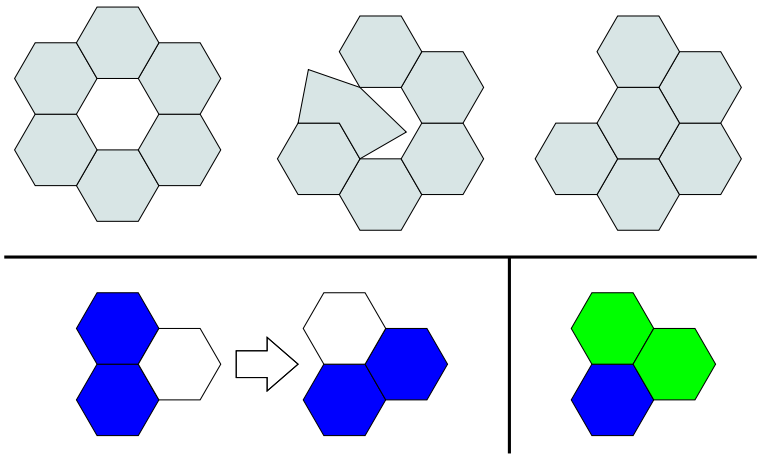


FIGURE 3.5. Chirikjian et al.'s Hexagonal Metamorphic Robot. Top how the 6 bar linkage mechanism deforms during a movement. Bottom left, how the labels of the lattice change. Bottom right, Ghrist's diagrammatic notation for describing the local context required for a move to be valid, as moves are symmetric w.r.t time, it is enough to show the trace of the move in green, indicating where a subunit could move between.

mechanism allowing a subunit to deform its hexagonal shape in order to relocate to an adjacent location. As the details of how the mechanical implementation executes a move is not important from a planning perspective, a discrete planning model can be used, whereby subunits move to adjacent locations each iteration of time if the local constraints permit. The local conditions at some location (a rigid body transform) on the embedding space required for a move to be valid is shown in Figure 3.5 bottom left, as well as the resulting state of the embedding space. As moves are symmetric across time it is simply necessary to identify which locations are changing state. The local constraints then can be written as a motion catalog of move generators , shown in figure 3.5 bottom right, a graphical nomenclature penned by Ghrist *et al.* [25].

As well as building prototypes for metamorphic robots, the group developed planning algorithms for the system that for shape-to-shape reconfiguration tasks. Pamecha and Chirikjian developed a planning strategy for the Hexagonal Metamorphic Robot (HMR) built upon simulated annealing [58]. Their variety of simulated annealing started at the start configuration, and took moves that improved a local heuristic function, or chose a random move if no improvement could be found until either the goal was found or a well defined maximum moves bound was exceeded. Their simulated annealing algorithm was prevented from back tracking to the immediately previously seen state as a speed optimization. They provided two strict base metrics for use as the guiding heuristic function, the overlap heuristic and the optimal assignment heuristic. They also made composite heuristics from the bases by linearly weighting, which was found to improve the performance of the heuristic for different classes of tasks. The optimal assignment heuristic cost $O(n^3)$ to compute at each iteration [8], but it is clear that even on some basic cases of reconfiguration tasks the overall time was growing exponentially with the number of subunits.

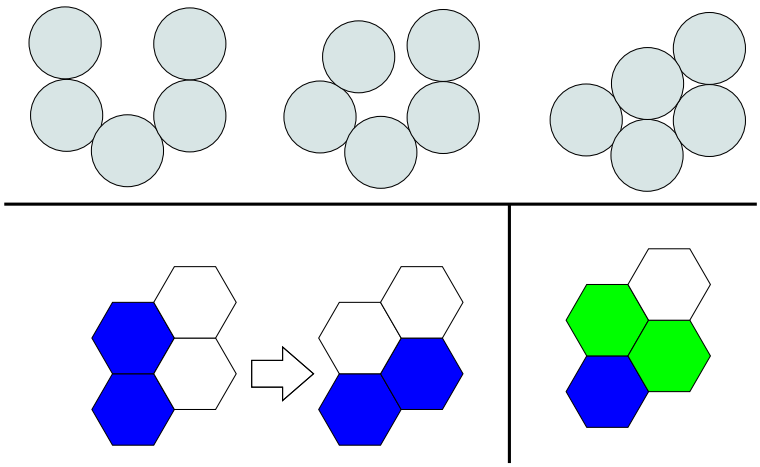


FIGURE 3.6. Top, The Claytronics platform 2D circular subunits roll over one another. Bottom left, the subunits can be viewed as operating on a hexagonal lattice, where locally a stationary neighbor muse be present to roll around, and space to accommodate the move transiently. Bottom right, in Ghrist's graphical nomenclature.

Walter *et al.* develop a planning algorithm for a modified version of Chirikjian's HMR [77]. In their model a unit can only move to an adjacent space if it, like Chirikjian's, had a neighboring subunit to pivot around, and, unlike Chirikjian's, had space opposite the pivot location. In this thesis we will refer to this motion model for the HMR as the Claytronics HMR motion model because the Claytronics project build a hardware implementation adhering to these very same constraints [27]. The extra space opposite the pivot simplified the hardware from a 6 bar linkage mechanism in Chirikjian's prototype, to a rigid circle (Figure 3.6 top) for the Claytronics. As we will see in Chapter 4 although the hardware implementation is simpler, the extra space makes the planning task more difficult.

Fitch and Butler propose in [20] propose solving task-to-task parallel motion planning using analytical techniques borrowed from reinforcement learning. In particular, they used a dynamic programming methodology to solve the parallel motion path planning problem. The convergence time for the dynamic programming step is dependent on the task geometry, and it is not possible to guarantee the task will be solved at all.

Zack Butler et al. [7] developed an motion algorithm for a cube based lattice SRS. Novel to the method was that it was not a motion planning algorithm to change a given shape into a desired shape, instead, it moved a connected collection of subunits in a desired direction. With just 5 local rules operating in parallel, Butler et al. elegantly proved that the aggregate would move in the desired direction, without individual subunits ever needing to know the underlying global structure of the aggregate. As no global planning algorithm was involved, it can be argued that the planner was of $O(1)$ time and space complexity that achieved a planning task without recourse to specifically describing the necessary shape to achieve the task. They also argued in the paper that the planning method was not tied to any particular model of SRS, because the 3D cubic model

used could be defined in terms of meta-module motions that could be implemented by different SRS motion catalogs. In chapter 6 we will specifically provide a mathematical notation for describing when one SRS planning algorithm can be instantiated on another.

The 3D cube model was used as a lattice motion model of reconfiguration for subsequent work by Fitch *et al.* that solved shape-to-shape reconfiguration tasks [21]. They reused many ideas from the MeltGrow algorithm originally designed for unit-compressible modules, but a key difference was that this algorithm was designed for labeled subunits. Their method was called the MeltSortGrow algorithm. Like the MeltGrow, mobile units were reconfigured into a linear chain, without checking the labellings. Then the 1D chain was sorted, in $O(n^2)$, and then melted into the desired labeled shape. As the original MeltGrow has a time complexity of $O(n^2)$, the sorting step did not disturb the worst case bounds. Unlike MeltGrow, where meta-modules could be tunneled through the structure, the 3D cube model did not contain compressible modules so subunits moved around the perimeter. In later work [22], a sub-procedure was added that permitted a subunit to tunnel through the structure by moving units out of the way and returning them to their original position. This increased the worst case complexity bounds $O(n^4)$, but allowed the algorithm to operate in amongst dense obstacles without the need to reconfigure via a 1D straight chain.

Tunneling can be seen as a method for ensuring a subunit is able to move regardless of the current geometry. Støy and Nagpal approached this issue in quite a different way. They constrained the geometry of the configuration to be a 3D scaffold shape[72]. The separated units into two classes, scaffold units and mobile units. Scaffold units were motionless, whose position adhered to the scaffold structure. They were able to sense if their should be another scaffold unit adjacent to them according to the goal, and could dictate to nearby mobile units to traverse into the location. The nearest mobile unit was guaranteed to be able to traverse to the location, because the local scaffolding structure implied it had the correct local conditions for its motion catalog, providing there were no other mobile-units in the way. The motion could not be blocked by another mobile unit though, because the recruited subunit was the *nearest*. So the process of convergence could be guaranteed. Thus a global scaffold could be constructed in any shape, from any initial conditions. The main drawback of the approach was that the goal configuration had to adhere to the scaffolding shape [73].

As part of the Claytronics [27] project's planning efforts, and interesting and novel stochastic method for SRS shape planning was developed by De Rosa *et al.* called shape sculpting via hole motion [17]. This approach did not attempt to determine the precise placement of every subunit, rather instead aimed at moving the units within an approximate boundary of a target shape. The motion constraints of the SRS were the Claytronics HMR. The volume of any shape was maintained to be "porous" by intrusions of enclosed space (holes). The holes were spontaneously created or destroyed at the perimeter of the shape, and wandered internally via Brownian diffusion dynamics. The existence of mobile holes uniformly within the structure meant that the perimeter of the shape could be moved by adjustment of the rate of hole removal/creation at particular boundary areas. However, only toy examples of reconfiguration were presented, and it remains to be seen whether

this method can be adjusted to handle complex shape reconfiguration tasks in a computationally efficient manner.

%CHANGE1%

Walter *et al.* produced a number of algorithms designed for reconfiguration planning for the Claytronics HMR motion catalog. These planners determined valid plans in linear time complexity, in a distributed manner using just local information, and were designed for parallel execution. However, these algorithms were only applicable for specialized classes of the more general motion planning space. In [76] plans were formed for a serial chain of HMR units to envelope an obstacle, with some loose restrictions on what shape the enveloped object could be. In [78] plans were formed between a serial chain to a different serial chain, and also between a serial chain and a restricted set of branched chains. While the efficiency and distributed nature of the algorithms are desirable, their main drawback is that they do not cover a large proportion of the potential motion planning state space of the HMR.

3.3. **Chain Type.** In chain type architectures, the subunits that make up the robot have internal degrees of freedom, joints, that alter the relative positions of the connection mechanisms. This creates two separate planning problems that need to be addressed. First, control of the internal degrees of freedom to achieve a pose. Second, chaining connection mechanism events to alter the kinematics of the system. The second problem actually requires the first to be solved because, in order to align connection mechanisms for a docking event, the configuration needs to be posed in some way. Existing literature largely follows these two major problem classes. The first problem, posing the SRS, has been addressed primarily to explore the locomotion abilities of chain type robots, with significant successes [70, 69]. The second problem, the core shape-to-shape planning problem has not been solved satisfactorily for any chain type SRS instantiation.

As the shape-to-shape reconfiguration problem for chain-type SRSs is a composite problem it is more difficult than for in the lattice case. A single joint motion in a chain-type SRS may displace the absolute positions of several subunits. Consequently, absolute position based heuristics do not work well. To further complicate matters, chain type architectures can contain loops in the kinematics which make the effective degrees of freedom less than the actual degrees of freedom. Reconfiguration planning in the chain case has been shown to be NP hard [30].

Yim attempted to address the issue of kinematic loops with Hierarchical Substructure Decomposition [9]. In his example of an HSD implementation, two low level substructures were defined, loop and chain. A configuration was defined as an interconnected *tree* of the lower level primitives. Kinematic loops could be accommodated if the loop could be isolated inside the low level loop primitive. So the notation could not represent arbitrary configurations in general, in-fact, a valid interpretation of this work would be that the notation forced configurations to be of a bounded tree width. A reconfiguration algorithm was presented that ignores the issue of self-intersection and provides no time complexity details.

Later Yim *et al.* [80] develop a different kind of chain-type reconfiguration algorithm that guarantees a plan in $O(log(n))$ steps. However, it must be noted this algorithm assumes that

a subunit can arbitrarily be disconnected and reconnected anywhere else on the structure. The plans formed change from a current connectivity graph into another, without concern for the pose problem, in a sub-optimal number of connectivity changes.

For the ATRON chain-type SRS, a number of reconfiguration strategies were tried out. ATRON units are identical but only contain 1DOF, which means individual units cannot displace themselves significant distances without the coordinated help from neighboring subunits. One approach for ATRON was to allow meta-modules contain three units connected in series to spontaneously emerge as a single logical entity from the structure, to locomote, and then to merge back with the structure again [15]. There seems to be some granularity advantage of planning in this manner over fixed sized grains as in the Melt-Grow algorithm [66], because the system as a whole does not have to be axis aligned into a course lattice, but it appears that some low level detail in the algorithm could not be determined and arbitrary shape-to-shape reconfiguration planning could not be demonstrated. Further investigations tried different meta-module definitions [15, 4], but convergence to a desired shape could not be guaranteed for any meta-module definitions tried. It is worth nothing that the shape-to-shape planning experiments were driven by a Euclidean absolute position distance based heuristic which does not map to the problem well.

Further work to investigate whether the choice of the planning algorithm affects reconfiguration shape-to-shape success for the ATRON was carried about by Brandt *et al.* [3]. A* [67] and a modified RRT-connect [38] algorithms were compared. Both algorithms performed poorly, with an exponential amount of time required to get to a desired shape as more modules are added. RRT-Connect was marginally more successful than the A* algorithm, but both were far from satisfactory. However, again the use of an absolute distance based heuristic was probably the true reason why both algorithms performed poorly. RRT-Connect is more tolerant of a bad heuristics than A*, because it emphasizes random exploration, which explains the results somewhat.

Reconfiguration planning for chain-type robots has not been solved adequately yet, no good heuristic has been found that can accommodate both the connectivity planning problem and the physical constraints of a particular system. Few algorithms have been tested at arbitrary shape-to-shape planning, and the few that have, take an exponential amount of time as more subunits are added.

3.4. **Hybrid.** Hybrid SRSs architectures aim to gain the advantages of lattice type SRSs and Chain type SRSs without the drawbacks. Hybrid SRSs are characterized by the units being able to be packed inside a regular lattice, which greatly aids physical alignment issues for docking/undocking units, as well as being simpler to plan with (reduced pose problem than for Chain-type reconfiguration planning). However, subunits, like their chain type counterparts, still have internal degrees of freedom, so are able to leave the lattice's regularity; enabling them to locomote like chain type SRSs. Thus, to a degree, the composite reconfiguration problem of Chain type SRS planning is decoupled for hybrid SRSs.

Arguably one of the most successful series of SRSs is the hybrid SRS the M-TRAN (I, II and III) robot [39, 55, 41]. Early attempts for solving the planning problem used genetic algorithms

[83] for sequencing connectivity changed to solve the reconfiguration problem. The scalability of the methodology is dubious. For locomotion, the team used a different evolutionary algorithm to generate central pattern generators (CPGs) that could be run in a distributed manor upon the robotic hardware [84].

Evolutionary techniques, like other global optimization, suffer from the curse of dimensionality so are unable to scale to SRSs containing 30 or more units. Ostergaard *et al.* developed an alternative technique for controlling M-TRAN that avoids the the scaling issue. Their techniques involves building shapes from concatenations of a few basic hand designed structures. Using only local information, individual M-TRAN have an hand crafted action policy that is able allow the global system to exhibit an interesting emergent behavior. One example they presented was a 1D snake, climbing up steps, which utilized sensor information to detect the steps corners. However, while it is clear that the examples they presented were self-reconfiguring in a scalable manner in response to the environment, it remains to be seen whether the software controllers can by synthesized by machine in a scalable manner as well [56].

3.5. **General Theory.** While most theoretical work to-date has focused on developing planning strategies focused on a particular SRS, there have been a number of notable works whose insights are not tied to a specific architecture of SRS. Unsurprisingly, general SRS conclusions have been drawn at a slower pace than SRS specific counterparts. Clearly several observations have to be made in different SRSs before a broader general conclusion can be confidently elucidated.

Butler *et al.* [6] observed that a specific planning algorithm could be instantiated on different SRS architectures. The planning algorithm was completely decentralized and local; subunits followed a predefined set of local rules that depended only on immediate neighbors states. Using local rules only, the aggregate was able to remain connected, and move in a global direction whilst navigating obstacles. The state of the system was represented on a 2D square lattice, so the planning algorithm appears tied to a square lattice based SRS. However, the general aspect of the work was realizing that drastically different SRSs, such as M-TRAN, could be packaged up into meta-modules, and that these meta-modules could be designed in such a way so that they moved according to the square lattice motion constraints required. Thus, Butler *et al.* developed a lattice based SRS algorithm, and showed that many different SRSs can be adapted with meta-modularization to exhibit a set of desired motion constraints. However, the process to adapting an SRS, just as regular meta-modularization, requires grouping subunits into atomic planning units and restricting motions, thus significantly reducing the state space utilized.

Ghrist and Peterson developed a very general language for describing a large class of SRSs, which has far reaching implications to the SRSs that it applies to [25, 1, 26]. A *local metamorphic system*, was described as having three elements: a substrate lattice, $\mathcal{L}$; an alphabet, Σ; and a set of generators Φ. Generators describe relabeling operations that can be applied to local areas of the lattice in order to change the state of the SRS (the motion model). Many SRS in the literature are not completely local because further global constraints are required to describe the systems motion models properly. For instance, Chirikjian's hexagonal metamorphic robot, in addition to Φ, must

also have an addition global constraint that the subunits remain connected. However, Ghrist does provide an example of a local version of the HMR which is very similar (by adjusting Φ) that does not require this check. So although many systems don't meet the local metamorphic definition initially, often an adjustment of the generators produces a similar SRS nearby that does.

The importance of Ghrist and Peterson's work was that systems that can be described using just those three ingredients imply certain properties will be present in their state space. They represented the state space of a system as a cubic complex. Each cube represented a set of underlying applications of the motion generators that could be applied in parallel without interference. Using tools from geometric group theory, Ghrist and Peterson showed that this state space complex was of non-positive curvature (amongst other things). This showed that a path through the state space could be locally deformed, in $O(n^2)$, into a time optimal path, of the same homology, using subunits moving in parallel. This implies that for a very broad range of SRSs, it is sufficient to solve the planing problem by considering just a single subunit moving at a time, and then convert it into a multi-move plan afterwards. It is for this reason that throughout this thesis we concentrate on planning tasks where only one subunit moves per iteration of time.

While Ghrist and Peterson provide a very general categorization of a large class of SRSs, if we view an SRS as a modular system, further broader principles may apply. Lipson asked whether we can define the meaning of modularity, regularity, and hierarchy for modular systems? [49] The motivation was that modular systems that exhibit these principle tend to easier for evolutionary processes to design scalable solutions for. Lipson concluded that **modularity** is the restriction of functionality to atomic building blocks. If functionality is manifest in a fitness function, then a subunit's contribution towards fitness will be independent to another's. This can be qualitatively measured checking the off diagonal elements of the Hessian matrix of the fitness function are small. **Regularity** can be qualitatively measured by measuring the compressibility of of the system, perhaps through minimum description length or Kolmogorov complexity. **Hierarchy** in modular terms is the ability for the system to be composed recursively. If a graph represented the permitted connectivity of elements, systems that are highly hierarchical, Lipson notes, will have the distribution of shortest paths adhering to a power law.

Modular systems that are highly **modular**, **regular** and **hierarchical** are preferable for evolution systems to design with for several reasons. A highly **modular** system suggests optimization can take place on parameters, in turn, rather than on the entire joint space in parallel. This transforms a high dimensional optimization into a series of low dimensional ones with obvious computational benefits. A **hierarchical** system suggests that there is a partial ordering of functionality, which again suggests an ordering, and associated reduction of computation complexity, of optimization is possible. **Regularity** suggests higher order correlations between components, which suggests that dimension reduction techniques can be applied successfully. As you can see, these three principles reduce the search space for an evolutionary process which is why these qualities imply a scalable planning methodology may exists.

Although Lipson designed these criteria for modular system, we can reinterpret them for an SRS motion planning context. Instead of using an evolutionary process to fit together modular pieces

for a task (fitness function), the evolutionary process can be substituted for a stochastic planning attempt between a desired start and goal configuration. In a highly **modular** SRS, the analogy to Lipson's concept would be that each subunit of the SRS could be moved between its start and goal location individually, without consideration of its neighbors. However, this property is not apparent in the SRSs studied here, as subunits remain connected at all times and so some form of coupling is always present. Lipson's **regularity** does have existing parallels in the literature though; the process of meta-modularization groups subunits into single planning elements which effectively reduces dimensionality. The motivation of meta-modularization is not to increase regularity though (although it does) but to increase mobility, which effectively reduces the coupling between planning units *i.e.* meta-modularization increases the **modularity**. It is the property of **hierarchy** though that may have the most interesting manifestation in the SRS planning context. Lipson's measure of hierarchy through fitting a power law is not directly helpful, but the link between partial ordering and hierarchy is. Partial ordering that would permit a planning problem to be solved recursively, perhaps by decomposition of the planning problem into composable solutions. Our analysis of reconfigurations state spaces in chapter 6 and subsequent conclusions revolve around this idea.

An ideal motion planning system for an SRS would be able to synthesize a planning algorithm for any SRS model provided, and would fully reflect the nuances of each motion model in the plans it produced. SRS motion planning problems are special cases of high dimensional planning spaces. They are special because each individual subunit of the system obey the same local motion model as other subunits of the system. This means that curse of dimensionality does not seem to apply, evident in the existence of many efficient (quadratic and sometimes linearly time bound) algorithms for SRS planning. However, so far there is no general algorithm, each algorithm is strongly coupled to SRS architecture it is designed for.

Ghrist and Peterson provide a concise definition of a local metamorphic system of which we shall borrow significant language from. In their work they discovered that local metamorphic system state spaces are of non-positive curvature. This suggests that a large class of SRS state spaces have certain identifiable properties which do indeed make them attractive from a planning perspective. That work, however, does not approach the problem of forming plans in the general setting.

4. General Motion Planning Methods for Self-Reconfiguration Planning [44]

The SRS community as a whole are in agreement that motion planning is a difficult problem, remarking that the problem bears the hall mark of a NP-hard problem [82], which was later proved to be the case for chain architectures [30]. The SRS literature contains many specialized planning algorithms targeted for specific instantiations of different robots, which scale differently as the number of units in the configurations grow. Unfortunately each specific algorithm is typically strongly coupled to the SRS instanciation it was designed for, so re-use of planners across different robotic models is impossible. An ideal motion planner for the SRS community is an algorithm that would scale near linearly in time complexity with the number of units in the configuration (and in near constant time using parallel distributed computation), and would be applicable to any SRS robot model.

Modern general modern motion planning techniques meet the criteria of not being coupled to a particular robot model, and the open question addressed in this chapter is do they scale well with the problem complexity? There has never been a comprehensive survey into the applicability of general motion planning methods to this problem domain as far as we know. Pamecha *et al.* applied simulated annealing (SA) to the hexagonal metamorphic robot [58], but this work was conducted before the discovery of efficient modern probabilistic sampling planning techniques, such as rapidly-exploring random trees (RRT-Connect) [47, 38] and probabilistic roadmap planning (PRM) [32]. Brandt compared greedy search with an RRT-Connect derivative for use with the ATRON SRS [3] and concluded RRT-Connect was better, albeit both scaled poorly. ATRON is a hybrid SRS however, so it is not clear whether these results transfer to lattice type architectures. In fact, it is an open question whether two SRS models from the same architecture family are comparable at all.

Greedy search uses a heuristic to evaluate the most promising state to expand from all previously seen. If the heuristic is poor however, greedy search tends to repeatedly expand states near each other as they tend to have a similar inaccurate heuristic values. With a bad heuristic, greedy search tends to dwells in areas of high heuristic error. Modern sampling based planners, in contrast, emphasize exploration of the state space, and are bias toward expanding states far away from previously seen states. Thus, they have found to scale better with lower quality heuristics because they do not dwell in patches states with high heuristic error.

In this chapter we apply a variety of general planners to two different models of a lattice based SRS. The first model is the Claytronics set of motion constraints for the hexagonal metamorphic robot and the second model is a refinement of the first as suggested by Ghrist [25]. Both models operate on a hex lattice and superficially have very similar motion constraints. It is of interest to see how slight changes in a model's motion constraints affect the ability planners to determine valid plans. We perform experiments using greedy search, RRT-Connect, PRM and SA. Greedy search provides a good classical planning benchmark to compare the modern sampling based planners, RRT-Connect and PRM. SA is included so that Pamecha *et al.'s* [58] original work can be compared in identical experimental conditions.

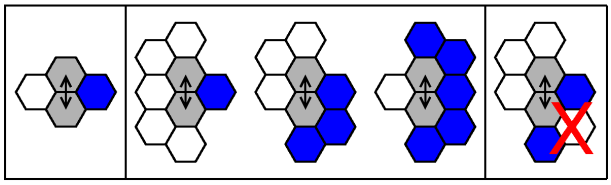


FIGURE 4.1. Catalog of permitted moves less isomorphisms. White denotes empty space, gray the moving component, and blue the other necessary robotic components. Left, for the Claytronics model, note that the moves on the right can only be applied *iff* all robotic components of the system remain connected. Middle, for Ghrist's model of the hexagonal metamorphic. Right, a potentially permitted move by the Claytronics model but not for Ghrist's

The planning task studied here is single move reconfiguration planning. Only one subunit in the system moves per iteration of time, and the task is to find a sequence of permissible moves that deforms a starting configuration into a desired configuration. The multi-move reconfiguration problem, where several subunits move per iteration of time, is addressed in more detail in later chapters.

4.1. **Models.** Two alternative formulations of the hexagonal metamorphic robot are investigated. Both SRSs are comprised of n robotic components located at points on a hexagonal lattice. At each iteration of time, one robotic component can move to adjacent lattice location subject to certain constraints, unless it is denoted as an anchor component in which case it can never move.

In the Claytronics model, a component at location a can move to an empty adjacent location b by pivoting around a common robotic neighbor to a and b. The other only common neighbor to a and b must be empty to allow space for the component to pivot (figure 4.1 left). During the move, the SRS should not become disconnected, so the moving component must not be a cut vertex *w.r.t* the connectivity graph. Ghrist's formulation is more restrictive, moves are not permitted that can change the global topology of the robot (such as introducing a loop), which implicitly implies all of the components remain connected. So for both systems all components remain connected, but in the Claytronics model this must be enforced using a global constraint, whereas in the Ghrist formulation this is implicitly enforced by the (local) motion constraints.

The state of a robotic configuration can be represented as a set of component locations and a set of anchor locations. We implemented the sets for the state representation in the simulator using *persistent* red-black trees [19]. A new configuration state could be forked from an old configuration state by an incremental deletion and reinsertion into the component set, *without* altering the original state, in logarithmic time and space complexity (*w.r.t.* the number of components in the system). However, for the original hexagonal metamorphic model, identifying the cut vertexes requires linear time to compute whereas in Ghrist's formulation no connectivity check is required and computing the next set of moves after a move has been applied can be done in logarithmic time.

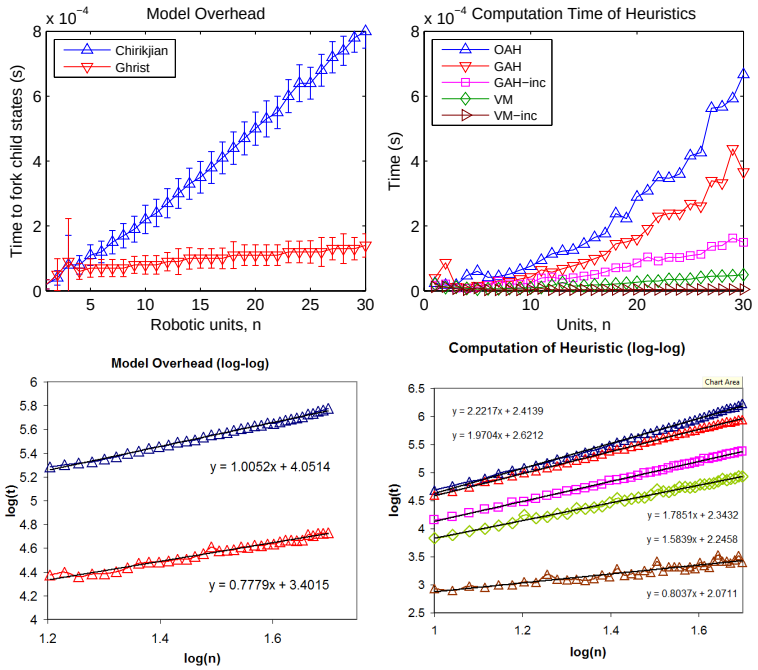


FIGURE 4.2. Left: The time taken for each model to generate valid children states. Right: Time to compute each of the heuristics, either from scratch or incrementally. In the bottom row the data has been smoothed and presented on log-log scale.

Figure 4.2 shows the computation time required to compute all the valid child states possible from a randomly generated state for the two models studied. In our implementations, determining the set of valid moves for a given state for the Claytronics model scales linearly with the number of subunits (gradient of 1 on log-log plot), whereas the Ghrist model scales sublinearly (0.8 gradient). While the Ghrist model is implemented by a logarithmic time complexity data structure, the number of valid moves on the surface scales at $\sqrt{n}$. It is of interest to test how the faster implementation of the Ghrist model affects overall planning time.

4.2. **Heuristics.** Informed general planning algorithms avoid evaluating the complete state space of a problem by obtaining guidance from a heuristic. We use a previously published heuristic used in SRS planning, the optimal assignment heuristic (OAH) literature, and developed two new heuristics aimed at improving computation time, the greedy assignment heuristic (GAH) and the vector map heuristic (VM).

In Pamecha *et al.'s* work for the hexagonal metamorphic robot they used a version of simulated annealing search [36] guided by a heuristic derived from the optimal assignment problem [58]. The optimal assignment heuristic, in this context, assigns each of the n subunits of the SRS to a unique goal location of the desired configuration of the SRS. The cost of an assignment is the unobstructed Manhattan distance from a subunit's current location to its assigned goal location. The Hungarian method finds the optimal pairings of subunits to goal locations such that the total cost is minimized,

at a computational cost of $O(n^3)$ [58]. This computational cost is paid each time the heuristic is called. The OAH heuristic was also used in Brandt's ATRON planning work [3].

The optimal assignment problem, in the context of the original combinatoric setting, is typically stated as how to assign m people to n jobs (one job to one person) optimally, given each person has a different aptitude for each job. The Hungarian method is performed on a $n \times m$ matrix of values, representing the ability of each person at doing a job. When used in the SRS context, the people are the current locations of the subunits and each job is to get a subunit at a specific goal location ($m = n$). The ability of each subunit's ability to get to each different possible goal location is approximated by the direct distance between the two locations. When expressed as a distance matrix, a column represent a location in the goal state, and a row represents a location in the current state. The distance matrix is filled with the lattice distance between the component's locations (figure 4.3, top). The OAH pairs all components from the current state to unique components of the target state, such that the sum of the pair distances is minimal. Graphically this is equivalent to highlighting n elements in the $n \times n$ distance matrix such that only one element is highlighted per row and column, and the sum of the highlighted values is the OAH distance (which is minimized by the Hungarian method at a computational time cost of $O(n^3)$, and space cost of $O(n^2)$).

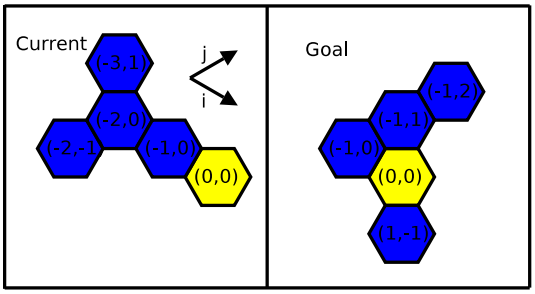
Given that we hope that an algorithm may be developed that is near-linear in time complexity, it is clear that the Hungarian method is computationally too expensive. The $O(n^3)$ time cost is exacerbated further in practical application because a general planning method will call the heuristic many times during operation. As the heuristic is only used as an approximation of the true state space to guide a search, we have developed a second heuristic that is a faster variant of the OAH called the incremental greedy assignment heuristic. The computation of an OAH like value is sped up by two optimizations. First, note that after a single move is applied only one row of the distance matrix changes, so calculating the value from scratch each iteration is unnecessary. Thus, the GAH calculates the value incrementally. Secondly, the requirement for an **optimal** assignment was removed and replaced with a good assignment instead, calculated greedily.

The GAH is implemented as follows. During a search to a fixed goal, the GAH maintains for each search state the distance matrix as n rows of n elements ordered by distance in n linked lists $r_1 \ldots r_n$. During single move planning, only one row changes per iteration, so a new ordered row can be calculated and sorted in $O(n)$ using radix sort.

When the GAH distance, *dist*, is requested, the algorithm initializes by removing the first element from each row list, r_i, and places the elements in a heap, H, which is also ordered by distance, at a cost of $O(n)$ (see figure 4.3, post init).

The algorithm then iterates: removal of an element $v_{r,c}$ from H. Placement of $v_{r,c}$ into "disposal" linked list d_r. If the element belonged to a row and column that both were currently marked as unsolved, then the row, r, and column, c, are marked as solved and *dist* is incremented by the distance value of the element v. Otherwise an element from r_r is removed and placed into H. Thus, once a row is marked as solved no more removals from the relevant r ordered link list will occur.

Task:



Distance matrix:

	$(1,-1)$	$(-1,0)$	$(-1,1)$	$(-1,2)$
$(-1,0)$	$2_{1,1}$	$0_{1,2}$	$1_{1,3}$	$2_{1,4}$
$(-2,0)$	$3_{2,1}$	$1_{2,2}$	$2_{2,3}$	$3_{2,4}$
$(-2,-1)$	$3_{3,1}$	$2_{3,2}$	$2_{3,3}$	$3_{3,4}$
$(-3,1)$	$4_{4,1}$	$2_{4,2}$	$2_{4,3}$	$1_{4,4}$

post initialization:

r_1	$2_{1,3}$	$2_{1,1}$	$2_{1,4}$	d_1	
r_2	$2_{2,3}$	$3_{2,4}$	$3_{2,1}$	d_2	
r_3	$2_{3,2}$	$3_{3,1}$	$3_{3,4}$	d_3	
r_4	$2_{4,2}$	$2_{4,3}$	$4_{4,1}$	d_4	

H				
	$0_{1,2}$	$1_{4,4}$	$1_{2,2}$	$2_{3,3}$

iteration 3:

r_1	$1_{1,3}$	$2_{1,1}$	$2_{1,4}$	d_1	$\mathbf{0_{1,2}}$
r_2	$3_{2,4}$	$3_{2,1}$		d_2	$1_{2,2}$
r_3	$2_{3,2}$	$3_{3,1}$	$3_{3,4}$	d_3	
r_4	$2_{4,2}$	$2_{4,3}$	$4_{4,1}$	d_4	$\mathbf{1_{4,4}}$

H		
	$2_{3,3}$	$2_{2,3}$

finished, $dist = 0 + 1 + 2 + 3 = 6$:

r_1	$1_{1,3}$	$2_{1,1}$	$2_{1,4}$	d_1	$\mathbf{0_{1,2}}$
r_2				d_2	$1_{2,2}$ $2_{2,4}$ $3_{2,3}$ $\mathbf{3_{2,1}}$
r_3	$2_{3,2}$	$3_{3,1}$	$3_{3,4}$	d_3	$\mathbf{2_{3,3}}$
r_4	$2_{4,2}$	$2_{4,3}$	$4_{4,1}$	d_4	$\mathbf{1_{4,4}}$

H	

FIGURE 4.3. GAH calculation example. Top row, calculating the GAH distance between a specific current and goal configurations. First row, the distance matrix between subunits of the goal configuration (columns) and the current configuration (rows). Elements of the distance matrix are sub-scripted with the row and column indices to aid in tracking them in the worked example. Post initialization, all rows have been sorted by distance and the first elements (smallest distance) have been placed in the heap, H. In this instance, the first two elements drawn from the heap are conflicted solutions for a unique row and column indices. So a distance for rows 1 and 4 are found first. The third element drawn from the heap, however, $(1_{2,2})$, conflicts with the already chosen element $0_{1,2}$ so is placed in the discard list but not used as part of the GAH final solution. Further elements are drawn from H and added to the discard lists until element $3_{2,1}$ is encountered in the heap. After $3_{2,1}$ all rows have an unconflicted distance associated, which are summed. The linked lists r can be concatenated to d to recover the original post initialization state.

The iterative procedure is terminated once all rows and columns have been marked as solved. At this point H will be empty, and the d linked lists will contain all the elements removed from the r lists, in order. The r lists are repaired to their original condition by prepending the d lists in $O(n)$ time. By storing references to the r lists in an persistent RB-tree, the rows data structures can be swapped, without disturbing the presorted data, in $O(log(n))$. Thus the GAH heuristic is *persistent* and *incremental*.

The overall time complexity of the heuristic depends on how many elements pass through H. In the best case, the first n withdrawals lead to a pairing at a cost of $O(n)$. The best case naturally occurs when the two compared configurations are identical, when each row and column contains a unique zero distanced element. In the worse case all n^2 elements must be added and removed from H at a cost of $O(n^2)$.

The final heuristic evaluated is the vector map heuristic. This heuristic converts a configuration into a real valued vector. Two configurations, a and b with vector representation v_a and v_b are evaluated by the VM heuristic as having a distance equivalent to the Euclidean norm of $v_a - v_b$.

The vector representation has k elements.

$v = [v_1, v_2 \dots v_k]^T$

The values of v are calculated from k hex coordinates, $l_1 \dots l_k$, that are distributed over the embedding space. The value of v_i is calculated by:

$$v_i = w_i \sum_{\forall r \in robot} e^{-y_i d_{hex}(l_i, r)}, y > 0$$

w and y are free parameters that will be optimized later. The negative exponent implies v_i is high only when the majority of the robotic mass is near hex coordinate l_i.

Optimal placement for the l locations we have not determined, but intuition suggests that the points should be concentrated towards the anchor node where most robotic mass is likely to be located. In the placement procedure presented here, measurement points are placed incrementally up to the number of components n. 6 points are placed equidistant to one another in concentric rings around the anchor. The placement ring increases in radius by one each increment of the algorithm. In each ring, the 6 points are equidistant, but their exact placement is found by maximizing their nearest neighbor distance to already placed locations (Fig 4.4). So the length of a the vector representation of a robot with n units is $6n + 1$

Good values for the constants vectors w and y were chosen by optimizing a cost function, J, derived from OAH distances on training examples.

$$J_{w,d} = \sum_{j,i \in train} (OAH(i,j) - VM_{w,d}(i,j))^2$$

%CHANGE 4A%

The VM heuristic can be calculated from scratch in $O(n^2)$ and the vector can be updated incrementally in $O(n)$. The VM heuristic can be presented to a search as *persistent* and *incremental* at a time cost of $O(n)$. The overhead factor associated with calculating the vector representation

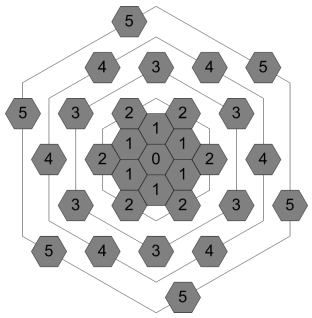


FIGURE 4.4. Placement of kernels in embedding space, for up to $n = 5$

of a configuration is very low. The effect of adding/subtracting a subunit at a specific location, q, on the vector representation v, is to add/subtract a constant vector, c_q, to v given by:-

$$c_i = w_i e^{-y_i d_{hex}(l_i, q)}$$

As the size of the embedded space only grows at $O(n^2)$ (being a 2D space), we cached values of c lazily as subunit moved around in the search space. So when a subunit moved from position s to position e, the vector representation could be updated very quickly incrementally using $v_e = v_s - c_s + c_e$, where c_e or $c_s$could be retrieved from a cache if already encountered before.

A consequence of the VM heuristic construction is that the vector representation is a function of a single configuration. So if the goal configuration changes during search, unlike the GAH, the vector representation of the search nodes can continue to be updated incrementally. In contrast, the OAH and GAH heuristics are functions of a pair of configurations. The motivation of the VM heuristic was to speed up nearest neighbor queries (NN) which are key components of RRT-Connect and PRM planners, where the goal changes frequently in the subsearches.

The time complexity of each of the heuristics was empirically determined by computing the estimated distance between randomly generated configurations, figure 4.2. The results of empirically estimating the time complexity of the heuristics differ slightly from theoretical expectation. The OAH is expected to have cubic complexity but has been empirically observed with a lower exponent value, specifically $O(n^{2.2})$. We believe the unexpected efficiency is due to the distance matrices of two configurations tending to contain more row-column unique 0 elements as n grows. 0 elements cause logical shortcuts in the code execution. Empirical observations of computing GAH from scratch met expectations of quadratic computation time, and the expected computational gains were also observed when computing GAH incrementally; $O(1.8n)$. The VM was expected to cost $O(n^2)$ to compute from scratch but beat expectation in empirical experiments with a run-time of $O(1.5n)$, furthermore the VM heuristic when calculated incrementally also beat $O(n)$ theoretical expectation by running at $O(0.8n)$ in empirical experiments. We attribute VM's empirical gain in speed due to the effect of lazily caching some of the calculations at a memory cost of $O(n^2)$.

4.3. **Planners.** Greedy search was implemented in a standard way, as outlined in [67]. In brief, greedy search operates by ordering seen states by their estimated distance to the goal, estimated

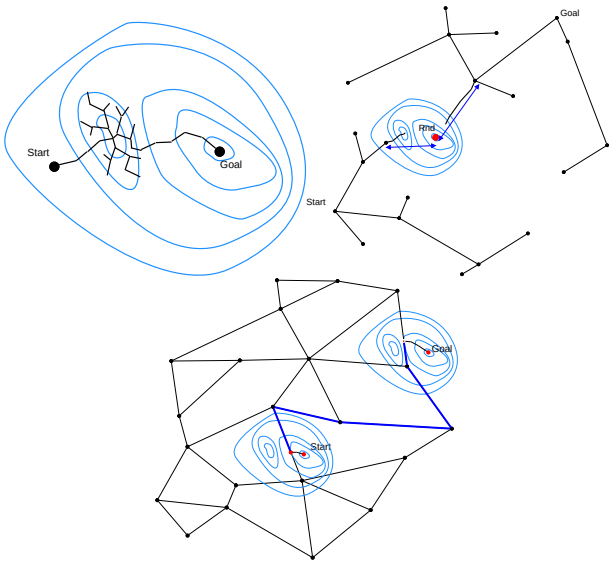


FIGURE 4.5. The differences in the way the different planners find paths between start and a goal states. Light blue contours denote the gradient of the heuristic. Greedy search (top left) expands states that are estimated to be near the goal, this means states in local minima must be enumerated before progress continues toward the goal. In RRT-Connect (top right), ϵ length branches are grown from trees rooted at the start and goal configurations towards randomly generated points in the space, using a local search. Local minima only damage the performance of a single local search toward an individual randomly generated location. In PRM (bottom) a coarse roadmap (graph) is pre-generated. When a plan is requested, the nearest road is typically a short distance away from the start and goal states, so the impact of local minima on a local search towards nearby roads is greatly reduced.

by the provided heuristic. Each iteration the state that appears nearest to the goal is expanded by adding the neighboring states into the queue. The process is repeated until the goal is discovered. The algorithm is initialized by adding the start state into the queue. It is equivalent to an A* search where the current distance from the starting state to a current state is ignored, which reduces the number of states evaluated compared to A*, but loses the optimality of A*. An RB tree data structure was used as the primary queue.

If the heuristic does not match the true distance between states, the search will expand states in a local minima. Because nearby states will have a similar heuristic value (the heuristic should be smooth), the search will repeatedly expand nearby states until all states in the local minima are expanded. This is shown diagrammatically in figure 4.5 and this effect is one of the motivations for sampling based planners.

The first sampling based planner for evaluation was the RRT-Connect algorithm. We refer the reader to [38] for a detailed description of its implementation from which we did not deviate.

RRT-Connect grows two trees from the start and goal configuration towards randomly generated states. Each iteration, a new state is randomly generated, and the nearest vertex of one tree is used as starting state for a local search toward the random state using a supplied heuristic. The local subsearch (we used Greedy search) is only run for a distance of ϵ. Wherever the local search terminates in the space, the other tree is then grown towards that state in a similar manner. If the trees connect, then a path can be found between the two tree roots. We set $\epsilon = 20$ for the extend and connect subsearches. We found, in agreement with [38], that the algorithm was not particularly sensitive to the value of ϵ. Our implementation was coded such that a different metric could be used for answering the nearest neighbor (NN) queries than the metric used to guide the subsearches.

RRT-Connect is rapidly exploring because the state space is expanded from the nearest previously seen state toward randomly generated states. If the randomly generated state is drawn uniformly from the state space, the chance that a previously expanded state happens to be the nearest is proportion to the Voronoi region. Thus states that are near unexplored regions of the state space are the most likely to be expanded further, until the algorithm has good coverage of the state space. This means RRT-Connect is less sensitive to the quality of the heuristic used compared to greedy search. A poor performing heuristic does not tend to prevent the algorithm from exploring, figure 4.5.

The second sampling based planner evaluated was PRM. Multiple search queries are performed after a single map construction phase. The purpose of the map is to approximate the overall state space using predetermined path segments, found using some other planner. These segments are joined into a graph. When a path through the state space is queried, the short distance from the start state to the nearest mapped road is planned to, and similarly the goal to the nearest mapped road. A complete path can then be found between the start state and goal state using a route from the roadmap and the two short dynamically found paths (figure 4.5). PRM tends to be much faster because a large proportion of the planning route is pregenerated, so most of the time is spend forming planning between the state and goal states to roadmap vertices.

As this study is focused on the time spent to form plans, we do not study the map generation part of PRM. For experiments we randomly generated n^2 waypoints and assumed a connected roadmap could be constructed from them. Our results look at how different PRM variants compare at planning a path from the start and goal states to roadmap vertexes. We compared greedy search and RRT-Connect as the subsearch methodologies. RRT-Connect also offers an alternative "connected to waypoint" strategy. As the PRM requires a path to any nearby waypoint, instead of attempting to find a path to the nearest waypoint, we tried instantiating a RRT-Connect subsearch with the two nearest waypoints as roots of the goal tree (now a forest). This multi-goal modification to the RRT-Connect algorithm affects implementation very little, but provides a choice of goal which may be useful if the nearest neighbor metric believes a waypoint is close but is actually difficult to plan to. The multi-objective RRT variant is denoted by RRT_MO.

We implemented Pamecha *et al.*'s simulated annealing (SA) algorithm as described in [58]. SA is a randomized walk through the state space, biased in the direction suggested by the heuristic.

Their SA implementation is notable for being greedy, uniformly selecting moves that lower the heuristics distance, and disallowing backwards moves. The temperature was fixed at 20.

Both PRM and RRT require random configurations to be generated during operation. We generated random configurations iteratively, with the initial configuration as a single anchor. In each iteration a robotic component was uniformly randomly selected for construction upon. A target adjacent location was randomly selected from the 6 possible sites surrounding the construction component. The constraints of the model were then checked to ensure valid configuration was constructed. For the Chirikjian model, this was simply checking the target location was empty. For the Ghrist model, loops cannot be introduced, so adjacent locations to the target were checked to ensure the proposed placement did not divide empty areas of space. Clearly these procedures do not sample uniformly from the set of possible configuration equivalence classes, but doing so is itself a challenging and open question. We describe this process in more detail in appendix 1.

4.4. **Experiments.** The experiments are presented as follows. First, each planning algorithm, greedy search, RRT, PRM and SA, is studied individually in order to ascertain the best performing variant with respect to total wallclock planning time using different heuristic and subsearch combinations. Wallclock time is used as the primary performance metric as it integrates the different time complexities and different number of states evaluated during a planning run into one practical comparable metric across different planning implementations. However, wallclock time is corrupted by noise from non-deterministic garbage collection arising from the java implementation. So in addition to wallclock time, the number of states evaluated by a planner is also presented, which does not suffer from garbage collection noise. Spikes in wallclock time can be cross referenced with states evaluated to determine whether the spike is caused by a garbage collection run, or by a particularly hard planning problem. After ascertaining the best combination of a planing algorithm and a guiding heuristic, the best performing combinations are compared side-by-side to provide a good view of what the best achievable is. Since PRM using an expensive precomputation of a roadmap, we also present the roadmap construction times separately.

A hundred reconfiguration tasks were generated for each complexity level, n, by generating pairs of random configurations containing n components. The configurations were generated using the Ghrist model's generator used in the RRT and PRM planners. The same set of tasks were used to test both the Claytronics model and the Ghrist model. Simulations were run on an Acer aspire 5630 laptop, 1.6Ghz Core Duo (but only one core was utilized) with 2GB of RAM. As a pragmatic necessity, if a search failed to complete within 60s the search was aborted, thus truncating the results. In many of the results presented the average wallclock planning times for for successful planning runs differ only slightly between planner and heuristic combinations, but significant difference can be seen when viewing the proportion of tasks that ran within the 60s time.

In the first experiment we compared three heuristics at guiding a greedy search. Figure 4.6 shows the performance on the two SRS models. For both models the VM heuristic clearly performs badly, being slower, evaluating more states and failing to complete more tasks in the necessary time than any of the other heuristics. VM is clearly not a good heuristic for guiding a greedy search.

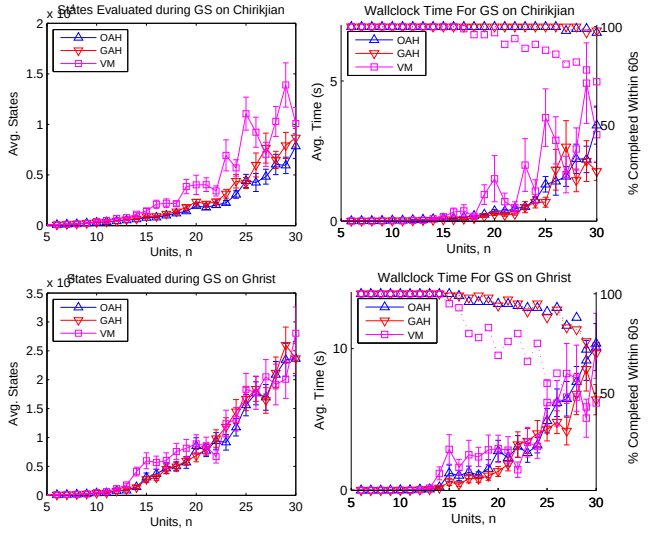


FIGURE 4.6. Comparison of heuristics for greedy search.

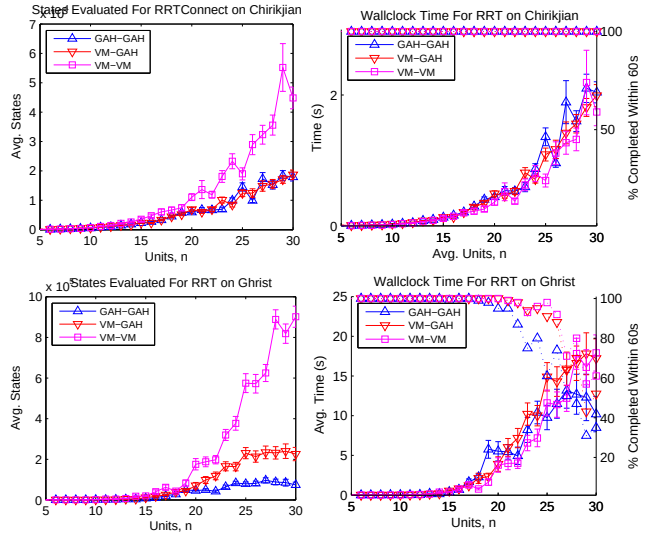


FIGURE 4.7. Comparison of RRT-Connect variants.

OAH and GAH perform similarly on both models. GAH evaluates more states on average than OAH, but tends to be quicker overall. The speed gains of GAH is more pronounced on the Ghrist model. This is probably due to a combination of factors, the Ghrist model's states are computed faster relative to the heuristics computation time, and overall solving the tasks using the Ghrist model is harder and requires significantly more states.

In the second experiment we evaluated RRT-Connect with different heuristics for the NN query and the subsearch. As GAH performed better than OAH in the experiments above, we discounted OAH for clarity. Figure 4.7 shows the performance of the RRT-Connect algorithm. Using GAH as both the NN metric and the local search heuristic, which is the most natural implementation, is

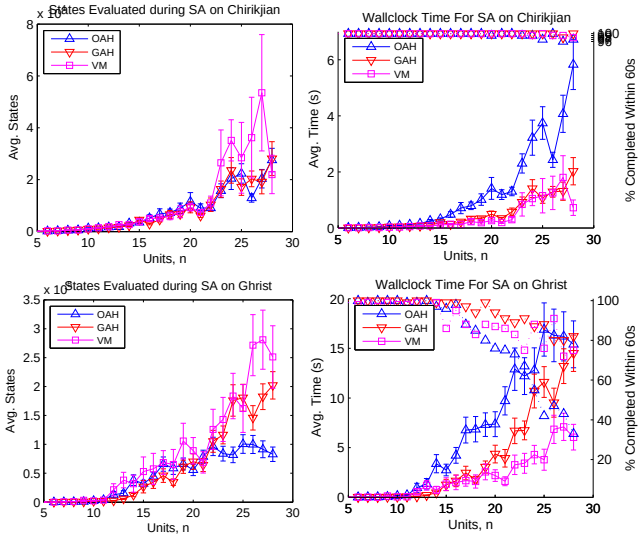


FIGURE 4.8. Comparison of heuristics for SA

the worst variant evaluated. The GAH-GAH combination is the slowest variant on the Claytronics model, and fails to complete the most tasks for the Ghrist model. Using the cheaper VM heuristic as the NN metric improves performance, which is seen clearest on the Ghrist model tasks.

Somewhat surprising is that using VM as NN metric and local search is overall the best RRT-connect variant. While this combination leads to a much large number of states evaluated, the sheer speed that the heuristic can be computed compensates. We note though that the number of states the algorithm is searching through is growing at an alarming rate. We speculate that at higher number of components the VM-VM RRT-Connect variants performance probably will start degrading faster relative to the other variants.

In the third experiment (figure 4.8) we compared the effect of the OAH, GAH and VM heuristics guiding SA. SA using OAH is clearly much slower on the Claytronics model, and fails to complete more tasks on the Ghrist model than the other heuristics. Between GAH and VM it is clear on the Ghrist model that VM fails to complete more tasks than GAH. On the Claytronics model they both take about the same amount of wallclock time to complete tasks and both complete almost all of them. However, for the VM heuristic the variance in the amount of states evaluated is far higher than for GAH, thus we conclude GAH is the best heuristic for guiding a SA search as high variability is undesirable.

We compared numerous variants for the PRM evaluation, see figure 4.9, by changing the sub-search variants along with the NN metric and guiding heuristics. The two PRM variants that used greedy search guided by GAH to get to the nearest waypoints are best at reducing the number of states evaluated. Of those two, the variant which used the computationally cheap VM heuristic as the metric to identify the nearest waypoints from the n^2 sized set overall had a lower average wallclock time.

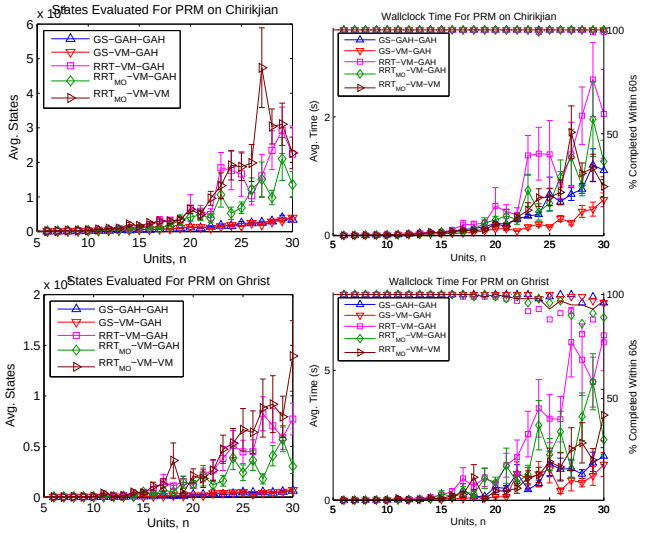


FIGURE 4.9. Comparison of PRM variants

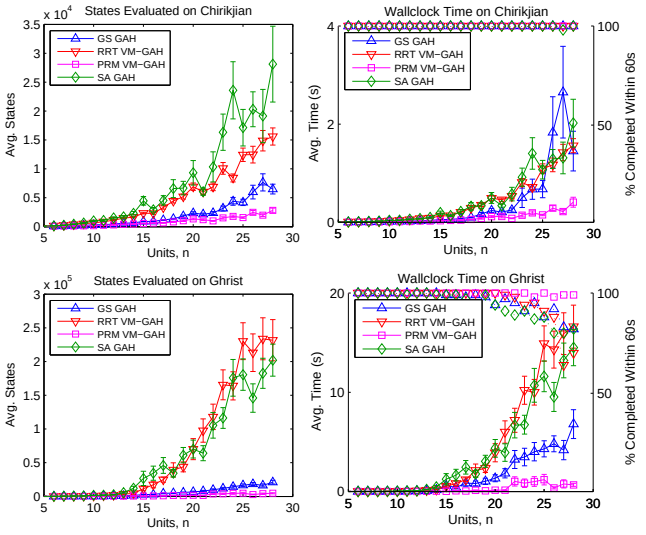


FIGURE 4.10. Overall comparison of planning methodologies

The two standard PRM variants which used RRT to plan to the nearest waypoints were the worse variants for PRM with respect to wallclock time. The multi-objective RRT searches that planned to the two nearest locations on the roadmap for a given start and end configuration performed better. So providing a choice of solutions for the RRT to find is a good strategy, as it provides a fallback option for the RRT if the nearest waypoint happens to be difficult to plan toward. The multi-objective RRT that used VM for the NN metric and search heuristic was the fastest RRT PRM variant studied, but still not faster than using greedy search.

We compared the best performing variants of each general search methodology (Figure 4.10). From conclusions drawn from the experiments above it is clear that VM is a better metric for

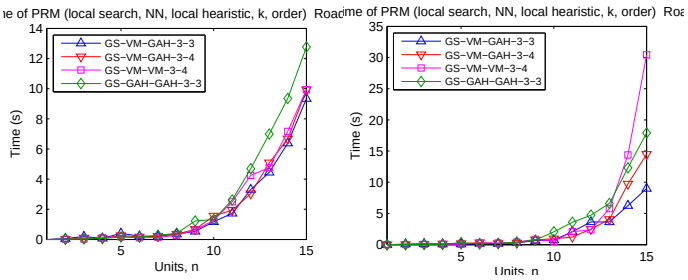


FIGURE 4.11. PRM roadmap construction cost

answering NN queries than GAH or OAH. As a local heuristic, GAH is better than OAH once computational time is factored into account. While we discovered that it might be better to use VM for NN and local heuristic for RRT-Connect we have chosen the variant of VM for NN with GAH as the local heuristic to represent the best variant in the side-by-side search algorithm comparison, as this aligns better with the other motion planning algorithm results.

Figure 4.10 summarizes the results across planners. SA and RRT stand out by evaluating large amount of states during searches. SA is slightly slower than RRT in wallclock computation time than RRT on the Claytronics model, and fails to complete slightly more searches on the Ghrist model. It is surprising that a bias random walk, SA, is even competitive with the more sophisticated RRT algorithm which uses a memory.

Of the single shot planners, greedy search clearly performs better on the Ghrist model, failing less and running faster and evaluating fewer states. This is also generally true on the Chirikjian model too, apart from some noisy cases.

The PRM planner is by far the best planning methodology considered for answering queries quickly. This is not surprising given the precomputation of a roadmap. We used just n^2 vertices which seems to be enough to prevent computation times scaling exponentially with problem complexity. Figure 4.11 shows the time it took for construction of a roadmap. It is common in PRM roadmap construction to first choose r nearest neighbors for each vertex and then to attempt to make k connections($k \leq r$) to the r neighbors. We found, in accord with our other results, a PRM using VM as the NN metric combined with greedy search as the subsearch with GAH as local heuristic to be the fastest search methodology for connecting roadmap vertexes. We choose $k = 3$ as this is the minimum order for a non-trivial roadmap, and we found using $r = 4$ does not improve performance over $r = 3$. Roadmap construction was expensive to compute.

4.5. **Discussion.** The vastly improved performance of the PRM planner over the single shot planners demonstrates that reconfiguration planning over short distances can be be done effectively using OAH to guide a subsearch. We only used n^2 waypoints in the roadmap, which was enough to prevent the initial local and goal searches scaling exponential with problem complexity. However, roadmap construction, does at least costs a high order polynomial if not exponential time to compute. Thus means that PRM is not a solution that could scale to thousands of subunits.

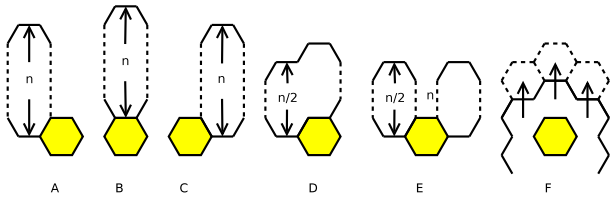


FIGURE 4.12. Some key configurations cases encountered during planning. See accompanying text in the discussion.

After computation time for state evaluations are factored into account, there is little evidence to recommend the RRT-Connect algorithm over greedy search or SA for planning for Claytronics's hexagonal metamorphic robot model. Furthermore, if one plans for the Ghrist model, then RRT-Connect is significantly worse algorithm than greedy search and SA. In this domain of SRS architecture, greedy search seems to be the best single shot planner.

Greedy search's key feature is that no state is evaluated twice. One must conclude that RRT's rapid exploration mechanism does not work very well in this domain. Consider the cases in figure 4.12. The distance between case A and case B is evaluated as n by the OAH, yet reconfiguration must go via intermediate case D which is at a distance of $n^2/4$ from both. Thus a local minima in encountered when planning from A to B, albeit a shallow one which greedy search will enumerate quickly. Going from case A to case C is a similar situation for the Claytronics model, but radically different for the Ghrist model. The intermediate case for A to C for the Claytronics model is case E, yet for the Ghrist model the units at the tips of the extensions are unable to bridge the gap, because doing so introduces a loop in the global topology of the SRS. In order for the Ghrist model to go from cases like A to C, some intermediate configuration whereby units are concentrated around the anchor, like in F, must be part of the solution path. If RRT-Connect is planning from case F through randomized exploration, it is equally likely that extension will occur in any direction. Once an extension waypoint is in RRT's expansion graph, it will naturally be more likely to be expanded again over expansion at F, because it becomes a boundary vertex in the Voronoi partitioning of the exploration space. If the initial random expansion from case F happened to be in the direction of case A when the true goal was in the direction of C, it will become increasing unlikely that a random generation of a state will force exploration from case F toward case C. Further, because of the Ghrist constraints, the exploration that will occur subsequently after initial expansion at A will not be able to create loops in order to ever reach C. The Ghrist model's state space, more than the Claytronics model's, seems to contain bug trap constructions[48].

Brandt concluded that for the ATRON model, RRT-Connect was a better planning algorithm than greedy search. The ATRON model is a closed chain type SRS. We have found those results not to be applicable for the lattice based SRS studied here. For Brandt, both algorithms performed poorly with 7 units in the SRS. Brandt used an OAH derivative heuristic to direct the planners. We attribute the poor performance of the algorithms to the heuristic not correlating well to the motion constraints. RRT-Connect in this case did better than greedy search because its performance is less dependent on the quality of the heuristic. Overall the OAH heuristic is a reasonable approximation

to the Claytronics model's state space, an adequate approximation to the Ghrist model's state space and a very bad approximation to closed chain SRS state spaces.

4.6. **Conclusion.** We are motivated in determining a general methodology for SRS reconfiguration planning. We have evaluated established general motion planning methods that have been used successfully in other problem domains. We have introduced two new heuristics which have improved the application of these methods.

Empirical experiments revealed that PRM is the best performing planner because coarse long distance planning has been precomputed. There is not strong evidence to suggest RRT-Connect is better than greedy search when the OAH or derivative heuristics are applicable. When the number of subunits is high, neither greedy search nor RRT-Connect are able to plan efficiently. Together these facts imply the OAH heuristic is only effective for coordinating the movement of subunits when they are nearly at goal locations. In other words, the OAH heuristic fails to capture the global aggregate behavior of subunits, but it is an adequate local approximation.

Our motivation was to develop methods that are both general (applicable for different SRS models) and efficient (scalable to large numbers of subunits). While the methods examined here were general, they were inefficient. For large numbers of subunits, PRM is an unattractive solution because of the expensive precomputation of the roadmap. The difficulties faced by modern sampling-based motion planning algorithms suggest that there is a need for for even better heuristics that capture the essential structure of this complex problem domain, perhaps involving multi-level strategies that better exploit the geometry and topology of the space.

5. An Efficient Algorithm for Self-Reconfiguration Planning in a Modular Robot [45]

Our goal is to find a method of reconfiguration that is both general (applicable to any SRS) and efficient (linear in time complexity). In the previous chapter we evaluated a variety of modern general planning methodologies and found they did not scale well to modest numbers of subunits. In this chapter we will relax the self-imposed generality constraint and develop a algorithm that is designed to be as fast as possible for a specific set of motion constraints. After development of a fast algorithm we will look for patterns in the implementation that could potentially be generalized.

While there have been efficient planners developed elsewhere in the literature, no-one has yet developed a near linear time algorithm for the hexagonal metamorphic robot. Furthermore, many efficient strategies have concentrated on meta-modularization which greatly reduces the state space by imposing a coarse high level structure to the geometry of subunits. Meta-modularization, in essence, is a wrapper for one raw state space, $\mathbb{R}$ so that it is presented as another (easier) meta-modularized state space, $\mathbb{M}$. While this means that $O(k)$ algorithms that were applicable for $\mathbb{M}$ can run on a specific subspace of the $\mathbb{R}$ space, much of $\mathbb{R}$ is lost in the process, and the planner for the $\mathbb{M}$ state-space cannot possibly utilize all useful motion paths in the $\mathbb{R}$ space.

Rus and Vona were the first to apply the technique of meta-modularization. It was for a different class of SRSs, those that are comprised of unit compressible subunits. In their work, units were assigned into small groups of virtual elements that lay on a coarse embedding lattice space [65]. A dictionary of moves was developed operating on the the coarser abstraction and planning was simplified. The overall time complexity for their planning algorithm was $O(n^2)$.

The original motion catalog for the Chirikjian HMR, and the Claytronics HMR both suffer from the explicit enforcing of the constrain that subunits remain connected, at a cost of $O(n)$. Ghrist's catalog does not need this check, so this catalog was used as a starting point to create the most efficient algorithm possible. We add further constraints to the Ghrist model in order to create a new catalog of moves for a HMR designed for efficient long range planning, we call this the surface model, whose state space we denote with $\mathbb{S}$. We denote Claytronics state space with $\mathbb{C}$. We design two shape-to-shape planning algorithms; first, an efficient surface-to-surface planner; and secondly, a slower Claytronic-to-surface planner. By utilizing both planning algorithms, the broader class of Claytronic-to-Claytronic shape-to-shape reconfiguration tasks will be solvable as in figure 5.1. As the Claytronic-to-surface portion of the plan will be the smallest piece, the efficiency of this part of the planning process will contribute less to the overall efficiency of the entire planning process.

Chirikjian and the Claytronics catalogs both require global information about the connectivity to check if a candidate move is valid. Global information affects computation efficiency, but also hampers distributing the planning computation. The surface model, on the other hand, can be enforced using local information only, and perhaps suggests the model may be useful not just for being fast for global planning, but also as a useful subspace for designing a distributed planner.

5.1. Surface model.

The surface model is a further constrained version of the Ghrist model. The surface model's state space is denoted as $\mathbb{S}$ which is contained within the Ghrist space, $\mathbb{S} \leq \mathbb{G}$.

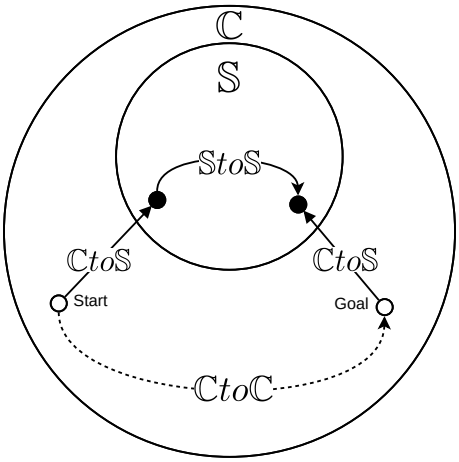


FIGURE 5.1. $\mathbb{C}to\mathbb{C}$ are formed by concatenating: 1, a$\mathbb{C}to\mathbb{S}$ plan from the stat state to the nearest $\mathbb{S}$ state; 2, a reversed $\mathbb{C}to\mathbb{S}$ plan from the goal to the nearest $\mathbb{S}$ state; 3, an $\mathbb{S}to\mathbb{S}$ plan between the two discovered $\mathbb{S}$ states.

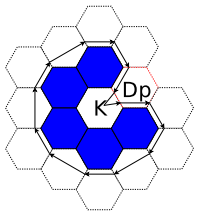


FIGURE 5.2. The Hamiltonian tour, around the surface for an example configuration. Robotic units are shaded blue. The two possible surface violations are shown. A kink is denoted K, and a dual path is shown as Dp. Both of these violations are not permitted in surface adhering configurations.

In order to describe the constraints imposed upon configurations belonging to the surface model, we require an additional data structure. While we describe this data structure in a language of global constraints, it can actually be enforced using information local to individual subunits (see appendix 2). So, a Hamiltonian tour around the adjacent external space is maintained using a spatially indexed linked list data structure (Figure 5.2). This data structure is represented by surface elements stored in a tree indexed by hex coordinates. A surface element is comprised of three pairs of *directional* pointers .

The first element of each pair of direction pointers represents the incoming direction of a Hamiltonian traversal visitation, and the second element represents the outgoing direction.

The ET is represented using indirect directional pointers, rather than absolute references to adjacent surface elements. This permits part of the tour to be rewritten without having to update all pointers. Therefore, the ET can be updated after a single-move at constant cost (figure 5.3).

In addition to the tour, several other statistics about the tour are maintained; a set of kink violations, and a set of dual path violations. A kink violation is the location of where a Euler tour

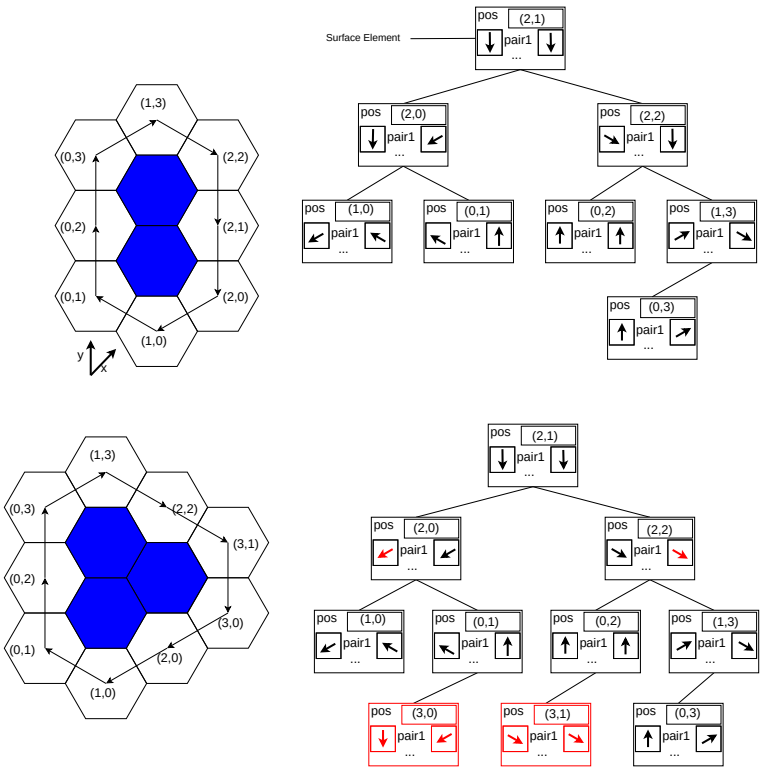


FIGURE 5.3. The Hamiltonian path around a configuration is stored in a tree of surface elements. When the Hamiltonian path needs to be updated, for instance when a subunit is added, this can be done by manipulating the spatial pointers in the immediate area. Thus, modifications can be done $O(log(n)$ and can be implemented persistently [19], using the ordering implied by the hex coordinates.

visitations enters and leaves through the same edge. A dual path violation is a location that is visited more than once on the tour. Access to the set of kink violations is denoted by the function $\_.K : \Sigma^* \mapsto set\,of\,\mathbb{P}$ and the dual paths by $\_.D : \Sigma^* \mapsto set\,of\,\mathbb{P}$

A configuration c is said to be a member of the unconstrained surface state space when there are no kink or surface violations i.e. $\mathbb{S} = \{c | c.K = 0 \wedge c.D = 0\}$. The motivation for this definition is to permit unconstrained motion for mobile subunits around the perimeter. If subunit can move, then by definition from the Ghrist catalog it is on the surface of the robotic volume. Prevention of a move by lack of pivoting space is impossible because no dual path violations implies there is always space around the surface. A change in global topology cannot occur because no kinks exist with which a mobile unit could bridge. Said another way, the surface constraints prevent 1 wide tunnels of space in the robotic mass (figure 5.4). It is only 1 wide tunnels that block peripheral movement in the Ghrist catalog. Thus, for the surface model, it is unnecessary to check the intermediate states when moving a unit from a location on the surface to another location on the surface.

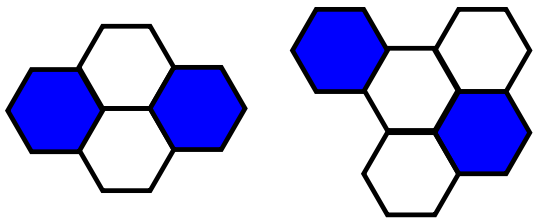


FIGURE 5.4. The cases of a 1 wide tunnel of empty space (less rotations). These figures were generated by a machine learning procedure described in appendix 2.

Algorithm 1 *LongMove* moves a long distance, and updates the Euler tour in constant time.

$$LongMove : \Sigma * \times (M_L) \mapsto \Sigma *$$
$$LongMove(c_s, (m_s, m_e)) \triangleq c_e$$
$$c_e \leftarrow ((c_s.R - \{m_e\}) \cup \{m_s\}, c_s.A)$$
$$c_e.ET \leftarrow UpdateEulerTour(c_s.ET, m_s, m_e)$$

For convenience we define a function *LongMove* which moves a robotic component on a configuration to anywhere, updating the ET as it does so (Algorithm 1). There is no validity checking in this function, but this is done elsewhere in the algorithms presented later.

Plans executable in the Claytronic reconfiguration state space can reach configurations belonging to the surface state space because $\mathbb{S} \leq \mathbb{G} \leq \mathbb{C}$. As moves are reversible (Figure **??**), *i.e.* $move(c_s, m) = c_e \models move(c_e, m^{-1}) = c_s$, a path from $c \in \mathbb{C}$ to $s \in \mathbb{S}$ can be reversed to find a path from s to c.

5.2. **Planning.** Our target is to determine a plan between two arbitrary Claytronics configurations. In this section we describe how surface-to-surface reconfiguration tasks can be solved efficiently. We then show how Claytronics configurations can be converted to surface configurations. The two forms of previous plans can be combined into one single-move plan. Finally, we describe how the single-move plan is converted to a multi-move plan.

As some of our algorithmic claims are empirically derived, we describe our empirical experiments here, with the relevant data interleaved with the algorithmic descriptions later. The data from first set of experiments was used to test the most efficient sub-planning stages. A 100 random configurations belonging to $\mathbb{C}$ containing $1000x$ units were generated for $x = 1 \ldots 20$. For the second set of experiments, 100 configurations were generated containing $100x$ units for $x = 1 \ldots 35$.

Random configurations were generated iteratively starting with a single anchor node. Mobile robotic nodes were added by uniform randomly selecting a component, then uniformly randomly selecting an empty neighbor (if one existed) and placing a unit there until the desired number of units were placed. While our results would be more representative of average case behavior if configurations were sampled directly from the state space of $\mathbb{C}$, this itself is an open and challenging problem [50]. The procedure and challenges faced are described in detail in appendix 1. We do not believe that the non-uniformity of the sampling procedure affects the average case complexity in a qualitative way.

Algorithm 2 Updating the labeling for $PLACED$ and $GROW$ is initiated at a location, loc. If the loc label changes to $PLACED$, the function recurs.

$$updateInc : \mathbb{P} \times \mathbb{S} \times \mathbb{S} \times (\mathbb{P} \mapsto L) \mapsto (\mathbb{P} \mapsto L)$$
$$updateInc(loc, c_{curr}, c_{goal}, labels) \triangleq labels$$
$$\text{if}(loc \in c_{curr}.U \wedge loc \in c_{goal}.U)$$
$$labels(loc) \leftarrow PLACED$$
$$\text{for}(\forall q.isAdj(q, loc))$$
$$updateInc(q, c_{curr}, c_{goal}, labels)$$
$$\text{if}(loc \notin c_{curr}.R \wedge loc \in c_{goal}.R)$$
$$\text{if}((c_{curr}.R \cup \{loc\}, c_{curr}.A) \in \mathbb{S})$$
$$labels(loc) \leftarrow GROW^{+}$$
$$\text{else } labels(loc) \leftarrow GROW$$

Algorithm 3 Updating the contraction labels is performed in patches of radius 2 around the location, loc. The $canMove$ function tests whether a given location on a configuration can move according to the Ghrist catalog.

$$updateArea : \mathbb{P} \times \mathbb{S} \times \mathbb{S} \times (\mathbb{P} \mapsto L) \mapsto (\mathbb{P} \mapsto L)$$
$$updateArea(loc, c_{curr}, c_{goal}, labels) \triangleq labels$$
$$\text{for}(\forall q.d(q, loc) \leq 2)$$
$$\text{if}(canMove(q, c_{curr}) \wedge q \notin c_{goal}.R)$$
$$labels(loc) \leftarrow CONTRACT$$

5.2.1. *Surface-to-Surface.* Surface-to-surface planning determines a set of moves that change from one surface adhering configuration to another. Surface-to-surface planning does not need to consider intermediate single-move motions.

The planning algorithm incrementally improves a current configuration c towards a goal configuration g. Locations in the embedding space are labeled from $L = \{PLACED, GROW, GROW^{+}, CONTRACT, \emptyset\}$(Figure 5.5).

A location labeled $PLACED$ denotes a robotic location that no longer needs to be considered in order to improve c. The anchors are considered $PLACED$ on initialization, and robotic units adjacent to placed units that are also found in the goal configuration are considered placed too. As the *StoS* planner never moves $PLACED$ units, the number of placed locations only grows. As placed units are adjacent to already placed units, the labeling of placed units are updated incrementally in the function *updateInc* (Algorithm 2) which is essentially a depth first search.

$GROW/GROW^{+}$ locations are where robotic units could be placed. As such they are: always empty, adjacent to locations labeled $PLACED$, and where robotic units are located in the goal. The $GROW$ labels are updated incrementally by *updateInc* (Algorithm 2), they reflect the next possible directions the $PLACED$ depth first search can travel.

$CONTRACT$ locations denote: locations where robotic units currently are that can move, and locations which are not occupied in the goal configuration. $CONTRACT$ locations provide a

Units, n	fails	trials	95% C.I. of P(fail)	
250	264	10000	.0233	.0297
500	7	10000	.0003	.0014

TABLE 1. Probability of *StoS* failing to find a motion path from a randomly generated start and goal configuration

supply of units for moving into $GROW/GROW^+$ locations. After a move is applied to the current configuration, some units local to the move's start and end locations may become mobile or lose mobility. Thus, updating the $CONTRACT$ labels is a local operation to be applied after the configuration changes, at constant time cost by the function *updateArea* (Algorithm 3).

The superscript + is appended to the $GROW$ label when the addition of a robotic unit at that location results in a valid surface configuration. Movement to $GROW$ locations only results in a valid configuration if the removal from the corresponding $CONTRACT$ changes the local context of the $GROW$ area. Thus, we prioritize consideration of movements to $GROW^+$ locations because it is likelier that the move will result in a valid surface configuration; and reduces the amount of movements considered in *improve*().

The *StoS* planning algorithm iterates until all units in the current configuration are $PLACED$. It improves the current configuration by moving units from $CONTRACT$ locations to $GROW$ locations (Algorithm 4). As units become $PLACED$ by an incremental traversal of the connectivity graph, the summed total time of all calls to *updateInc* is $O(n)$. The time cost of *improve* depends on how many movement attempts are rejected because they fail to result in a valid surface configuration. Further, the number of calls to *improve* depends on how many times the planning algorithm must iterate. Figure 5.7B, shows that empirically the number of times the planning algorithm iterates is approximately $k\sqrt{n}$. Figure 5.7A, shows that the number of $GROW$–$CONTRACT$ pairs considered by *improve* grows very slowly with problem complexity.

There are, however, cases where no improvements can be found and an error is generated (see Figure 5.6). If this occurs, an random intermediate configuration is generated for c_{start} and c_{goal} to be planned between. Failures become less likely as the number of units in the configuration increases (Table 1) and presumably become irrelevant with respect to time complexity as $n \to \infty$.

The overall wallclock time of planning, including failure resolution, is shown in figure 5.7D. Computation time scales linearly with problem complexity in normal circumstances, but spikes are apparent where the system failed to find a solution in one attempt. As discussed, these spikes become irrelevant to time complexity, asymptotically.

5.2.2. *Claytronics-to-Surface.* The Claytronics configuration state space permits the overall morphology to contain holes. These cannot be removed by the Surface move catalog, so the aim of the Claytronics-to-surface conversion step is to plan a set of moves that removes any holes from the initial and goal configurations. Holes are removed by finding tunnels linking all holes and excavating units inside these tunnels to safe locations elsewhere on the configuration.

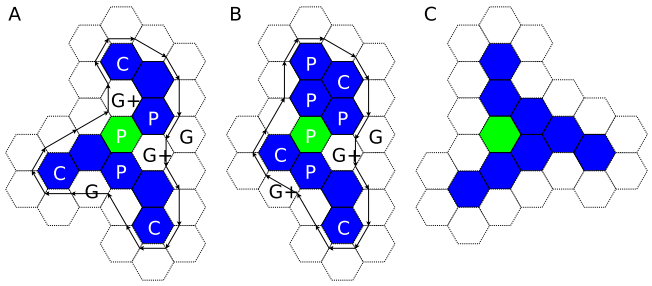


FIGURE 5.5. The labeling of $PLACED$, $GROW$,$GROW^+$, $CONTRACT$ and the Euler Tour for two configurations (A and B) planning toward a goal (C). B is one possible *longMove* option *improve* could suggest given A.

Algorithm 4 The 𝕊urface-to-𝕊urface planner.

$$improve : \mathbb{S} \times (\mathbb{P} \mapsto L) \mapsto \mathbb{S} \times \mathbb{M}_L$$
$$improve(c, labels) \triangleq$$
$$\textbf{for } \{s, e | (labels(s) = CONTRACT^+$$
$$\vee labels(s) = CONTRACT)$$
$$\wedge (labels(e) = GROW^+$$
$$\vee labels(e) = GROW)\}$$
$$\hat{c} \leftarrow LongMove(c, (s, e))$$
$$\textbf{if } (\hat{c} \in \mathbb{S})$$
$$\textbf{return } (\hat{c}, (s, e))$$
$$\textbf{throw error}$$

$$StoS\colon \mathbb{S} \times \mathbb{S} \mapsto M_L^k \times \mathbb{S}^k$$
$$StoS(c_{start}, c_{goal}) \triangleq (M, S)$$
$$c \leftarrow c_{start}$$
$$\textbf{for } \{a | a \in c.A\}$$
$$labels \leftarrow updateInc(c, g, labels)$$
$$\textbf{for } \{x | x \in c.R\}$$
$$labels \leftarrow updateArea(x, c, g, labels)$$
$$\textbf{while } (\exists l.l \in c.R \wedge labels(l) \neq PLACED)$$
$$((s, e), c) \leftarrow improve(c, labels)$$
$$labels \leftarrow updateInc(e, c, g, labels)$$
$$labels \leftarrow updateArea(s, c, g, labels)$$
$$labels \leftarrow updateArea(e, c, g, labels)$$
$$append(M, (s, e))$$
$$append(S, c)$$

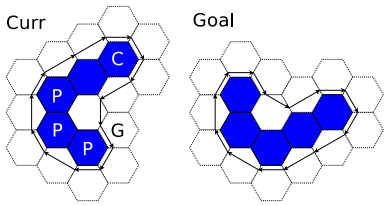


FIGURE 5.6. Example of an error situation. Moving the unit at the only *CONTRACT* location to the only *GROW* location (thus forming the configuration in Figure 5.2) causes a kink violation *i.e.* the Surface-to-Surface planner is stuck

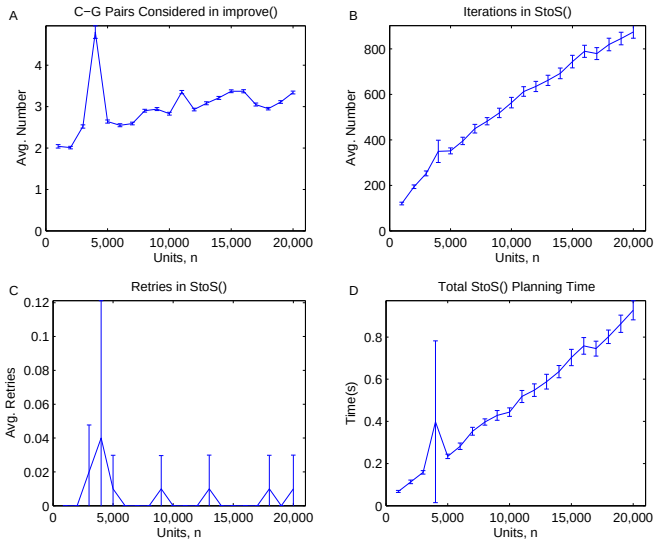


FIGURE 5.7. The Surface-to-Surface planner improves() routine finds a valid relocation of a subunit between a contract and growth location in very few tries (A), and the number of tries grows slowly with problem complexity (logarithmic?). A sub linear number of iterations are need to transform the stating configuration into the goal (B). The planner runs into difficulties, solving via a randomly generated intermediate, only at low problem complexities (C). Overall performance appears to be linear (D), spikes will occur less frequently as $n \to \infty$.

Firstly, it should be noted that from the moves catalog of the Claytronics model it is clear that components cannot move along the walls of an empty tunnel one space wide. So a two wide tunnel needs to be determined that links all holes. This is done in a two step process. In the first step a one wide tunnel is found and in the second step, the one wide tunnel is expanded to be two wide.

To find the one wide tunnel for a configuration, $c \in \mathbb{C}$, a connectivity graph $G_t = \mathcal{G}_{conn}(c.R \cup c.Adj)$ is constructed that includes the adjacent unoccupied location but not the anchor nodes. The edges are given weights by the (directed) function $w(x_1, x_2) \triangleq if(x_2 \in c.Adj)\ 0\ else\ 1$

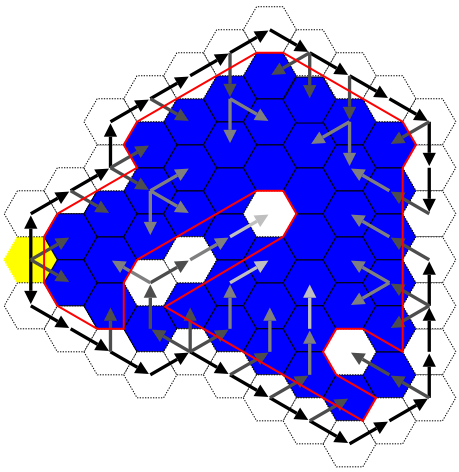


FIGURE 5.8. Top, the one wide tunnel procedure. The large arrows denote the order of Dijkstra expansion. The shade of the arrows indicate the geodesic distance from the external unoccupied start node (yellow). The red highlights the nodes in the minimal spanning tree that connects all unoccupied space. Bottom, the expansion of the one wide tunnel to two wide.

The weight function implies traversals into unoccupied locations on this graph "cost" nothing. To find a one wide tunnel we initiate a Dijkstra shortest path traversal initiated at an outside adjacent location. The traversal is terminated as soon as all unoccupied locations have been visited. The minimal spanning tree of the search graph containing all search nodes that expand unoccupied locations identifies the minimal set of units that connect all holes with a one wide tunnel (Figure 5.8). The time to perform the Dijkstra search is linear. Taking the minimal spanning tree is also linearly bounded. So overall determining what set of units lie within the one wide tunnel has a computation time of $O(n)$. We denote the operation as $tunnel_1(c \in \mathbb{C}) \mapsto R_1 \in \mathcal{P}(\mathbb{P})$.

The second step in the tunneling procedure is to expand the one wide tunnel to be two wide. Determining what side of a one wide section of tunnel to dig, in order to achieve the **minimal** set of nodes to remove, seems expensive to compute. Instead we chose a linearly bounded greedy procedure. First, all units of R_1 are removed from the configuration to yield a hole-free configuration $c' = (c.R - R_1, c.A)$. Then, every adjacent unoccupied location, $c'.Adj$, is checked to determine whether it is a one wide passage or a kink (figure 5.9). If it is, then the locally minimal set of neighbor units to undo its status are marked for removal, and c' is updated to reflect this (algorithm 5). Anchor locations cannot be part of the removal set. This procedure is named $tunnel_2$.

Robotic units to be moved are identified by $tunnel_2$ as a set R (figure 5.8). Removal of R from $c.R$ yields a surface configuration with which an unviolated ET, W, (described in the definition of a surface robot) can be wrapped around. W provides a "roadmap" to determine what surface locations are safe for robotic units to be moved to. The growth set, G, is defined as locations where placement of a robotic unit does not cause a violation in the ET.

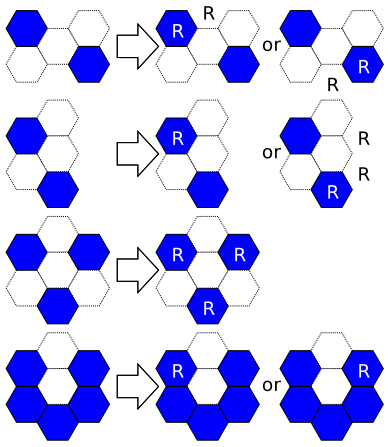


FIGURE 5.9. The one wide tunnel cases which need to be corrected in order to create a two wide tunnel. On the left is the matching context, on the right possible fixes, where R denotes which locations should be removed.

Algorithm 5 Expansion of a one wide tunnel, (Figure 5.8) is achieved by matching unoccupied adjacent space to problem categories (Figure 5.9). If a problem is found, the solution that requires the minimal amount of additional R tunneling units is selected as the local solution.

$$tunnel_2 : \mathbb{C} \mapsto set\,of\,\mathbb{P}$$
$$tunnel_2(c) \triangleq R$$
$$R_1 \leftarrow tunnel_1(c)$$
$$R \leftarrow R_1$$
$$c' = (c.R - R, c.A)$$
for $\{e | e \in A_{c'}\}$
if $(p \leftarrow matchProblem(e) \neq \emptyset)$
$$m \leftarrow \infty$$
for $\{s | s \in solutions(p)\}$
$$q \leftarrow |s \cap c'.R - R|$$
if $(q < m \wedge s \cap c.A = \emptyset)$
$$m \leftarrow q$$
$$s_{min} \leftarrow s$$
$$R \leftarrow R \cup s_{min}$$
$$c'.R \leftarrow c'.R - s_{min}$$

The planner solves the task by moving units at a location in R to a location in G. For speed, the planner tries to achieve these goals by using the Ghrist's motion catalog, and only uses the Chirikjian model's catalog when necessary.

The Ghrist catalog (a subset of the surface catalog) can be too limiting in some situations. The Ghrist catalog's motion constraints prevent a unit on the boundary of a hole from moving to open the hole (Figure 5.10). However, by definition, a hole is enclosed by a boundary of robotic units,

so the global topological change of opening the hole cannot disconnect the overall connectivity of the robot. While the Claytronics catalog would permit the move, it comes at a linear cost of a connectivity check. We can avoid this by filling all enclosed holes in the configuration with virtual robotic subunits, which will allow the move to be identified by the Ghrist catalog at constant cost. After the move has taken place, a virtual element will be in contact with empty space. All virtual elements in contact with empty space are removed recursively. As the Chirikjian-to-surface converter is only removing holes from the configuration, initial identification of virtual elements is $O(n)$ and the total time for removing all elements is also $O(n)$ (in much the same fashion of maintenance of the set P in the *StoS* planner).

To improve efficiency further, rather than trying to move elements of R out directly, a subset of R is maintained, R^+, which denotes those locations of R that contain units that can move according to the Ghrist catalog.

Figure 5.11A, shows the that the number of times the Ghrist catalog is used to remove elements from R scales as $\sqrt{n}$. In comparison, the number of times the Claytronics catalog is used scales very slowly, and in many problem instances is not needed at all (figure 5.11B).

The number of search sub-steps required to find a path for a unit from R to G, i.e. move a subunit out of a tunnel, appears to be independent of problem complexity (figure 5.11C). As the depth of tunnels is a 1D phenomenon, one might expect the depth to scale at $\sqrt{n}$ like the perimeter. This anomalous result may indicate the sampling procedure is not generating examples that are fully representative of the state space.

The algorithm only fails to remove a unit from R^+ when doing so disconnects the overall robot structure which, by definition, violates the Claytronics's motion constraints. Failure occurs when the tunneling procedure creates a tunneling structure that partitions the robot. So far we have only implemented an *ad hoc* partial solution to this problem. We allow the planner to initially solve as much as possible, and rerun the tunneling algorithm using added noise to the weight function in $tunnel_1$ when the planner gets stuck. This permits the tunneling procedure to try different tunneling structures. The planner retries tunneling a maximum of 10 times before deciding the conversion is impossible. Whilst this means some conversion attempts fail, the noisy tunneler does allow some extra configurations to be solved that initially couldn't. The majority of randomly generated configurations are solvable by this procedure (figure 5.11E). Retunneling $O(n)$ in time complexity, but the number of times it is required scales very slowly with problem complexity (figure 5.11D).

Retunneling only occurs when the planner gets stuck. Surprisingly, figure D is identical to B, and this has been thoroughly checked for validity. What this shows is that deferring to the Claytronics catalog after failing to move a subunit using the Ghrist catalog (figure B), is always immediately followed by a retunneling attempt. Using the Claytronics catalog for moving subunits is a redundant step, and could be omitted from future implementations.

The overall wallclock time to compute Claytronics-to-surface plans is shown in Figure 5.11F. The data suggest that, for most cases, the computation time is scaling linearly. There is an exception

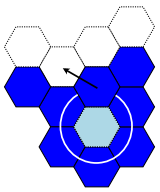


FIGURE 5.10. The Ghrist catalog is unable to open an enclosed hole. By adding a virtual robotic element to the configuration (light blue), a Ghrist catalog move becomes applicable. After the move, the virtual robotic element can be removed from the configuration. Calculation of holes, and placement of virtual elements, can be done once per *CtoS* call, which is computationally cheaper than multiple queries to the Claytronics catalog.

Algorithm 6 The Claytronics-to-Surface planner

$$C\_to\_S : \mathbb{C} \mapsto M_S^k \times \mathbb{C}^k \times \mathbb{S}$$
$$C\_to\_S(c) \triangleq (m, s, c)$$
$$m \leftarrow \emptyset$$
$$R \leftarrow tunnel_2(c)$$
$$W \leftarrow (c.R - R, c.A).E$$
$$\textbf{while}(|R| > 0)\text{:loop}$$
$$R^+ = \{r | r \in \mathbb{P} \wedge canMove(r, (c.R \cup holes(c), c.A))\}$$
$$\textbf{for } \{u | u \in R^+\}$$
$$\textbf{if } (p = \text{findAnyPathGhrist}(c, u, R) \neq null)$$
$$c, R, W, m, R^+ \leftarrow \text{update}(u, \text{end}(p)), \textbf{ goto } \text{loop}$$
$$\textbf{for } \{u | u \in R^+\}$$
$$\textbf{if}(p = \text{findAnyPathClaytronics}(c, u, R) \neq null)$$
$$c, R, W, m, R^+ \leftarrow \text{update}(u, \text{end}(p)), \textbf{ goto } \text{loop}$$
$$\textbf{if}(tries = 10) \textbf{ return } \text{error}$$
$$tries = \leftarrow tries + 1$$
$$R \leftarrow tunnel_2 noisy(c)$$
$$W \leftarrow (c.R - R, c.A).E$$

in an experiment with 12,000 units. We cannot explain this anomaly, as it does not correlate with any other spikes in the other metrics. It may be due to a Java garbage collection pause.

5.2.3. *Claytronics-to-Claytronics.* The Claytronics-to-Claytronics planner uses each of the planning algorithms described above as sub-steps to creating an overall single move plan to between arbitrary Claytronics configurations.

Given a start and end configuration, $s \in \mathbb{C}$, $e \in \mathbb{C}$, a single move plan is formed for each that transforms them into surface adhering configurations. A plan between these surface adhering configurations is found using *StoS*. This compressed plan contains *longMove*s, which is decompressed

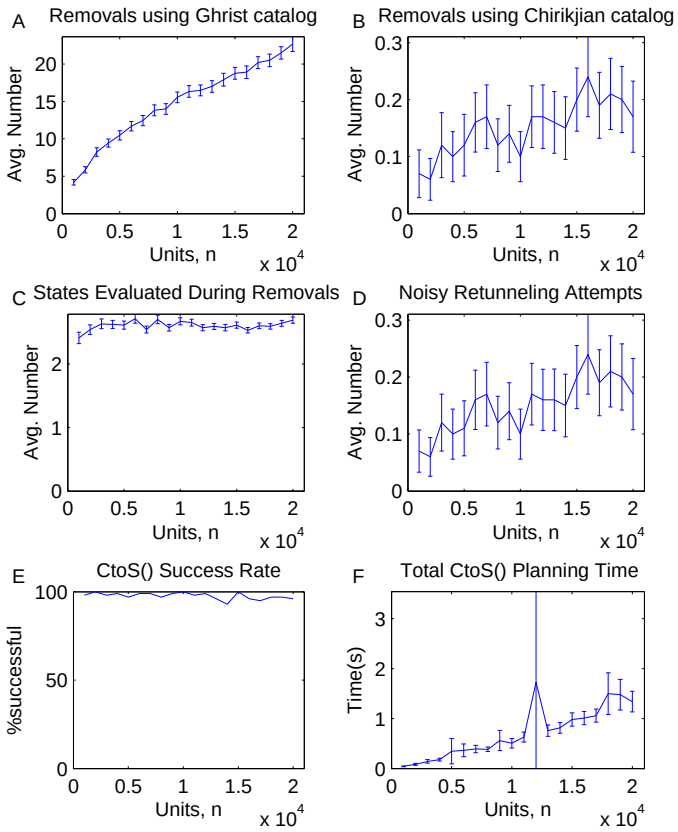


FIGURE 5.11. A Claytronic configuration can be converted to a surface configuration almost exclusively using the computationally $O(1)$ Ghrist moves (A) rather than requiring the $O(n)$ Claytronic moves (B). Subunits that needed to be removed did not seem to have to travel far, even in large configurations (C). Retunneling was not required often, but was required more as the complexity of the task increased (D). The overall success rate was very high, however, the failure rate increase with problem size (E). Overall planning time appears linear, apart from an anomalous spike which appears unrelated to other performance statistic gathered, so this is likely to be the Java garbage collection.

by searching the Ghrist state space. As *StoS* plans, on average, contain $\sqrt{n}$ long moves, which require $\sqrt{n}$ Ghrist moves to realize, decompression costs $O(n)$ (Figure 5.12A).

Finally, the separate plans are concatenated together to yield a plan that is a set of single moves that changes s to e.

5.2.4. *Single-Move to Multi-Move plan conversion.* The planning algorithms discussed so far achieve their aims by moving one component at a time. However, the time required to execute a plan on a hardware platform can be drastically improved if multiple units are permitted to move in parallel.

Ghrist provides [25] a method for converting a single-move plan into an optimal multi-move plan for moves using the Ghrist catalog. Once the single-moves plan is calculated, which contains

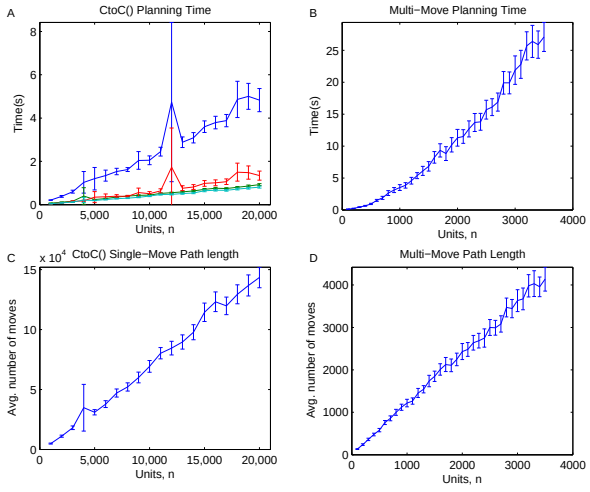


FIGURE 5.12.

moves from both Claytronics and Ghrist catalogs, Ghrist's multi-move conversion algorithm can be applied. The existence of Claytronics moves in the plan does not pose a problem. Ghrist's conversion algorithm is just applied to the solution sub-sequences that contain Ghrist moves.

The overall time to convert the single-move compressed plan to a multi move plan empirically appears to be bounded quadratic, in accordance with Ghrist's own analysis (Figure 5.12B). We note though, that when the multi-move conversion step is applied, if all intermediate states and moves are available then the conversion could be computed in a distributed fashion. So we propose representing the underlying state of configurations using *persistent* red black trees [19]. *Persistent* red-black trees can represent sets with all the normal operations taking $O(log_2(n))$ but modifications to the sets preserve the original versions at $O(log_2(n))$ in space cost. Using *persistent* red-black trees would increase the cost of computing a single-move plan from $O(n)$ to $O(nlog_2 n)$ but would permit all intermediate solution configuration states to be preserved. Then, it may be possible that a distributed version of Ghrist's multi-move conversion step could compute a multi-move plan in $O(nlog_2 n)$.

5.3. **Results.** The total time to form a single move plan from two randomly generated Claytronics configurations is shown in figure 5.12A. The total time to plan and convert into a multi-move plan is shown in figure 5.12B. Sometimes the planning algorithm can fail, caused by tunneling problems in the *CtoS* converter. The failure rate is shown in Figure 5.11E. On average, for over 97% of randomly generated tasks with up-to $20,000$ units, our algorithm finds a single move plan in linear time. While some of the *CtoS* computation times spike, the magnitude of the spike is still linearly bounded because only a finite number of $O(n)$ retries are attempted.

The average single-move path length is shown in figure 5.12C. The single-move path length scales linearly with the number of units in the configuration. The multi-move path lengths are shown in figure 5.12D. Moving units in parallel reduces the number of time steps necessary to change

from one configuration to another by several orders of magnitude. The multi-move plans scale sub-linearly with problem complexity, but not $\sqrt{n}$ as may be hoped.

5.4. **Discussion.** A common extension to the SRS reconfiguration planning problem is the addition of obstacles in the space. The algorithm presented here is easily extended to this situation. The dual path violation case can be generalized to indicate when an obstacle blocks pivot space around the surface of the robot.

The simple solution to obstacles, avoiding movement in the vicinity regardless of the current planning task, is reminiscent of a conservative approach to calculating a subset of $\mathcal{C}_{free}$ by taking the Minkowski sum of a bounding sphere with obstacles in the environment [14]. Our approach in *StoS* planning can be viewed as a utilization of this insight.

The SRS domain is difficult for planning because the robotic units are obstacles to each other. As we can control the robots though, this can be used to our advantage. Our approach has been to keep the surface of the configuration unconstrained for robot movement, in a sense, maximizing $\mathcal{C}_{free}$ for those robotic units who can move. This has enabled efficient planning over a large spanning subset of the Claytronics reconfiguration state space. In the following chapter (chapter 6), we will analyze the state space $\mathbb{S}$ further, to deduce why this state space is special in contrast to the underlying $\mathbb{C}$ space.

6. A Characterization of the Reconfiguration Space of Self-Reconfiguring Robotic Systems [46]

6.1. **Introduction.** An open goal of the SRS community is to developing a computationally efficient motion planning algorithm that is applicable to a broad range of different SRS architectures. In chapter 4 we saw that search based planning algorithms do not apply efficiently to the SRS domain. In chapter 5 we developed an *ad hoc* planning algorithm that was computationally efficient but specialized for a particular set of SRS constraints. The key to this motion planning algorithm was identification of an efficient subspace where the majority of planning could take place. In this chapter we pool knowledge from other efficient planning algorithms existing in the SRS literature, and the algorithm from chapter 5, to elucidate a rationale for why some SRS state spaces are more efficient that others. While this knowledge alone would not provide us with a general purpose motion planning algorithm for all SRSs, it could be a useful initial step in the construction of an SRS motion planning compiler.

We use the the Surface space as an example of an 'easy' state space that can be found within a number of possible 'hard' state spaces of the HMR. We demonstrate that the subspace is well connected (in a sense to be made precise), which is why planning tasks can be solved efficiently using greedy methods with a low probability of failure. We test this hypothesis by utilizing a sampling-based method to estimate quantitative descriptors of the algebraic connectivity of the state space. We compare the results from this specialized subspace against a more general model of HMR reconfiguration, and discover a striking qualitative difference in the behavior of the algebraic connectivity as the number of subunits in the configuration grows. The implication is that the Surface space contains few bottlenecks, even when there are high numbers of subunits.

A second desirable property of the Surface space is that the different instances of the reconfiguration space, corresponding to incremental addition of a subunit, are well ordered in a specific sense. Specifically, we prove that the reconfiguration graphs at increasing levels of complexity are ordered by the graph minor relation, in a way that seems to extend the notion of meta-modularization. Ordering by graph minors explains why certain SRS models can be solved recursively in a particularly simple and efficient way. We hope that these ideas might inspire further analysis of the global structure of reconfiguration spaces and algorithm designs.

While the specific results of this paper are phrased in the context of the study of a specific algorithm for a specific model of a SRS, the quantitative and analytical tools can be applied to any SRS, to explain when and why a subspace of a reconfiguration space for an SRS may be good to plan within, providing tools for characterizing and evaluating a subspace's suitability for efficient planning. In future work, we hope that these tools can be utilized to develop automated methods for identification of useful subspaces and other abstractions, to seed the development of SRS planning algorithms for different SRS architectures.

6.2. **Preliminaries.** Let $\mathbb{P}$ denote the set of points on a hexagonal lattice, $\mathcal{L}$. The metric $d : \mathbb{P} \times \mathbb{P} \mapsto \mathbb{Z}$ is defined as the Manhattan hex distance (see [12] for details). We say two locations, $x_1 \in \mathbb{P}$ and $x_2 \in \mathbb{P}$ are adjacent as $isAdj(x_1, x_2) \Leftrightarrow d(x_1, x_2) = 1$. The undirected connectivity

graph, $\mathcal{G}_{conn}$, of a set of locations, $V \in \mathcal{P}(\mathbb{P})$ ($\mathcal{P}$ denotes the power set function) is the graph constructed from $G(V, \{(e_1, e_2)|isAdj(e_1, e_2)\})$.

In all models of the HMR described here, a configuration, c, is a connected set of robotic subunit locations, $c \subset \mathcal{P}(\mathbb{P})$ where $\forall x, y \in c$ there exists a path in $\mathcal{G}_{conn}(Y)$. There are often further constraints to the admissible set of configurations depending on the HMR model.

A move, m, is an ordered pair of positions, $m \in \mathbb{M} = \mathbb{P} \times \mathbb{P}$. A single-move plan, is an ordered sequence of moves. Whether a move is admissible depends on the motion catalog, which is different for different models of the HMR.

We specialize the general definition of a metamorphic system by Ghrist *et al.* [1, 26] for describing HMR motion catalogs here. Ghrist *et al.* permitted an arbitrary alphabet of symbols to label an arbitrary embedding space to describe a specific state of the system. A 'state' in Ghrist *et al.* work is a configuration of subunits for our purposes. Our alphabet for labeling the hexagonal lattice, then, is simply $A = \{OCCUPIED, EMPTY\}$. By the Ghrist *et al.* definition, a local metamorphic system's permissible state transitions are completely described by a motion catalog, C, which is a collection of generators. A generator describes which labels may change (the trace) when a given context is present (the support). Specifically, a generator, $\phi \in C$ consists of a support, $SUP(\phi) \subset \mathcal{P}(\mathbb{P})$, a trace, $TR(\phi) \subset SUP(\phi)$ and an unordered pair of labeled local states, $\hat{U}_{0,1} : SUP(\phi) \mapsto A$ satisfying:

$$\hat{U}_0|_{SUP(\phi)-TR(\phi)} = \hat{U}_1|_{SUP(\phi)-TR(\phi)}$$

In other words the labeling of $\hat{U}_0$ and $\hat{U}_1$ are equal over the support locations, but may differ in the trace. For the HMR motion catalogs describe here, the trace consists of two adjacent locations, and the local states are labeled to reflect that a single unit moves from an $OCCUPIED$ location to an $EMPTY$ location.

Generators describe move classes, but an actual movement is carried out at a specific location in the embedding space. Ghrist *et al.* define an action of a generator $\phi \in C$ as a rigid body translation, $\Phi : SUP(\phi) \mapsto \mathcal{L}$, thus providing information as to where the generator was applied and in what direction. Given a state $U : \mathcal{L} \mapsto A$, the action is admissible if $\forall x. \hat{U}_0(x) = U(\Phi(x))$. The result of the action on the state is

$$\Phi[U] := \begin{cases} U & : on\, \mathcal{L} - \Phi(TR(\phi)) \\ \hat{U}_1(\Phi^{-1}) & : on\, \Phi(TR(\phi)) \end{cases}$$

In all the specific catalogs used here, the trace consists of two adjacent locations labeled $EMPTY$ and $OCCUPIED$ in $\hat{U}_0$ which are swapped for $\hat{U}_1$, representing a subunit moving to an adjacent location. So we can work out the rigid transform Φ from $m \in \mathbb{M}$. The Ghrist *et al.* notation is very general, and in the algorithms presented in this paper we only need to know whether a move is admissible or not, so for convenience we define the function $match_C : \mathbb{M}, \mapsto boolean$ to return true if the move is admissible for the given catalog, C.

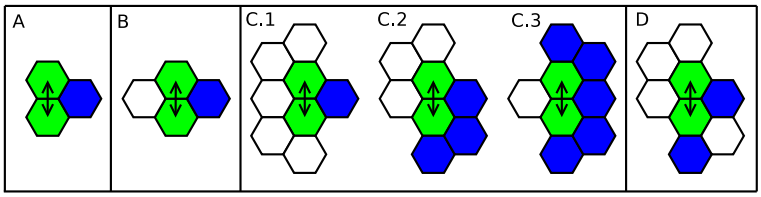


FIGURE 6.1. Previous motion catalogs modulo isomorphisms. Green denotes the trace, where a subunit can move between. Blue and white respectively denote where subunits must be/must not be in the local context for the move to be admissible. A: The original motion catalog for the hexagonal metamorphic robot by Chirikjian [12]. B: The motion catalog for the Claytronics prototype [35]. C: The three generators comprising of Ghrist's example motion catalog [1]. D: A move permitted by the Claytronics model but not Ghrist's as it changes the gross topology of the aggregate.

6.3. **Background.** The Surface space can be viewed as a further constrained version of Ghrist's HMR motion catalog. It inherits Ghrist's locality and admits a simple planning algorithm to solve Surface-to-Surface reconfiguration tasks efficiently. In chapter 5 we used the Surface space to form long distance plans within the Claytronics HMR reconfiguration state space, which allowed Claytronic-to-Claytronic reconfiguration tasks to be solved, empirically, on average, in linear time [45] for 97% of the state space. An arbitrary Claytronics HMR configuration, on average, is only a few moves away from a nearby Surface adhering configuration using the Claytronics motion catalog. It is thus tractable to compute a motion trajectory in the more general, and computationally less efficient, Claytronics space only a small distance to find a nearby Surface configuration. While the algorithm was highly efficient and operated on a large fraction of the Claytronics state space, the Claytronics-to-Surface planning step was essentially a heuristic method that had a failure rate proportional to problem complexity. In the interest of clarity, all the analysis in the following sections are restricted to the properties of the Surface state space, for which a well-behaved motion planner is presented.

So it is the nature of the reconfiguration state space of the Surface HMR motion model we wish to understand further. It is worth noting that the motivation for the Surface state space definition is removal of the planning difficulties associated with the Claytronics motion catalog requiring empty space opposite the pivot location. Previous attempts at abstracting away troublesome constraints in other SRS models have centered around meta-modularization of the state space [66]. In meta-modularization the atomic planning unit is actually a collection of SRS subunits (figure 6.2 A) with predefined sequences of moves that permit the meta-modules to move with fewer motion constraints than the underlying subunits (figure 6.2 B). The planning task is thus simplified, but at the cost of coarsening the embedding lattice significantly. The drawback of the methodology becomes apparent when one considers the proportion of configurations in the underlying SRS state space that have a representation in the meta-module state-space. Meta-module conforming configurations occupy an almost negligible proportion of the overall general state space. Meta-modularization, then, does not

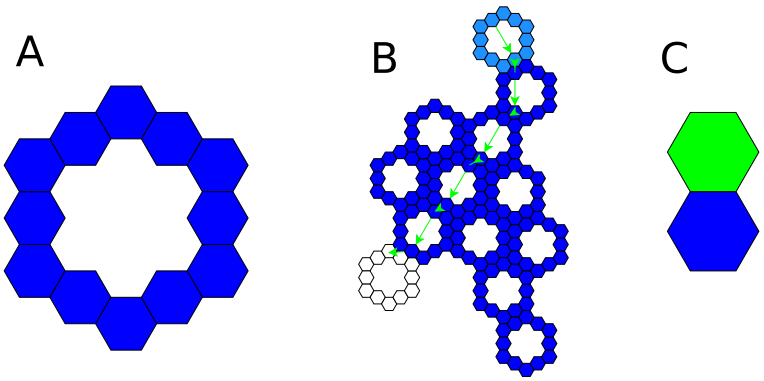


FIGURE 6.2. A: A potential meta-modularization of the HMR that permits mobility of the subunits found in the center of each hexagon edge using the Claytronics motion catalog, in particular, providing empty space opposite the pivot locations. B: A configuration built out of 12 such meta-modules, and an example of how local meta-module motion primitives can be daisy-chained together to the effect of moving one meta-module to an empty location adjacent to the perimeter. C: Expressing the local support required for a subunit to enter or leave a location.

lend itself well to being used as an intermediate path through a more general configuration space (for example, there are no meta-module configurations for 13 subunits for the example in figure 6.2). Both a meta-modularization subspace and the Surface subspace achieve similar qualitative behavior, simplifying planning, by adding additional motion constraints to an underlying motion model. The Surface space achieves a similar result to meta-modularization, but by sacrificing fewer configurations.

So the motivation for this work is to shed light on the question: why is it that adding extra constraints can sometimes make planning harder, as in adding constraints to the Chirikjian catalog to create the Claytronics catalog, and yet sometimes easier, as in adding constraints to the Ghrist catalog to create the Surface catalog, or by applying a meta-modularization strategy? By identifying general principals that describe when and how adding constraints can simplify planning, then we expect advances in reconfiguration algorithms for much harder models of SRS whose motion state space is difficult to mentally visualize *e.g.* M-TRAN [55]. Furthermore, adding artificial constraints to a model reduces the number of states the augmented system can express, so our work will aid in constructing constrained motion catalogs that sacrifice the minimum state space volume for a gain in planning efficiency.

6.4. **A Surface-to-Surface Planner.** A Surface adhering configuration, $c \in \mathbb{S}$, is defined as a configuration that permits a Hamiltonian path to be wrapped around the adjacent external locations ($adj(c)$). This requirement is compromised by two classes of violation. A kink violation is present when the peripheral tour leaves through the same edge it enters from (figure 6.3), and a dual path violation is where the tour traverses through the same location more than once (figure 6.3).

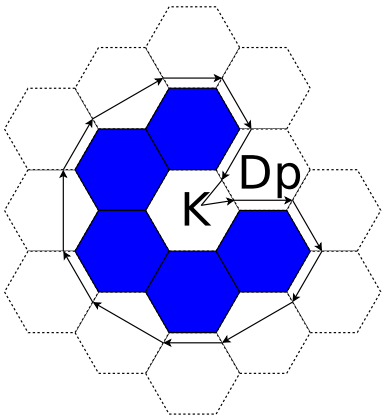


FIGURE 6.3. Wrapping a tour around a configuration. This configuration is not a valid Surface configuration because it contains a kink violation (K) and a dual path violation (Dp).

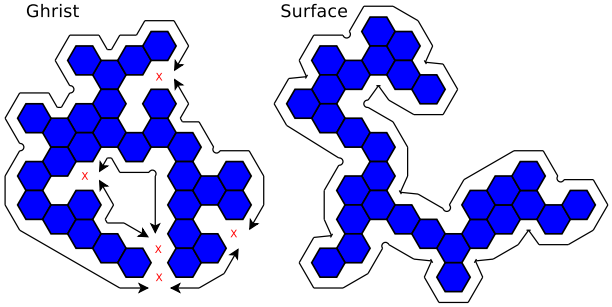


FIGURE 6.4. Examples of a valid Ghrist configuration and a valid Surface configuration. Ghrist configurations may contain narrow intrusions of space, which prevent subunits on the perimeter from crossing. Surface configurations, by construction, do not.

A valid Hamiltonian path implies several properties relevant to motion under the Claytronics and Ghrist motion catalogs. If an extra subunit is added, the subunit can move in a complete loop around the entire perimeter of the configuration. The lack of dual path violations implies there is always open space above the additional subunit for pivotal space. Absence of kink violations implies there cannot be a change of gross topology, specifically an introduction of enclosed space, caused by a subunit bridging the empty space adjacent to the kink (figure 6.1 D).

Note however, that while the added subunit is able to move around the configuration freely using Ghrist's motion catalog, it may not be able to stop anywhere on this path and still result in a valid Surface configuration. The moving subunit may itself cause kink or dual path violations. The Surface model's constraints merely imply that if a unit can be removed from one perimeter location and placed at another valid location, then a sequence of Ghrist motion moves will exist to link them (although violations may transiently be generated when executing the underlying Ghrist sequence).

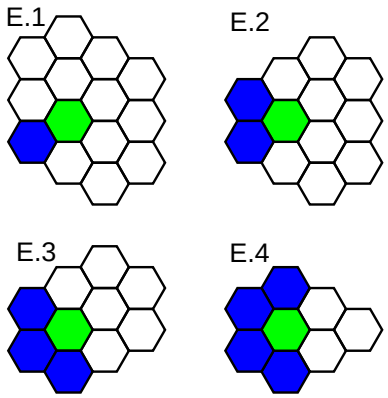


FIGURE 6.5. There are four different generators for adding/removing a subunit from a Surface configuration.

Whilst the Hamiltonian path constraint is a useful description of the Surface model's restrictions, both for implementation and visualization of the path around the configuration subunits take, we can rewrite this functionality in terms of a new set of local contexts for the motion catalog. This proves that the Surface HMR model is also a *local* metamorphic system by Ghrist's definition. A major difference with the motion catalog for the Surface model compared to the other HMR models is that the start and end locations for a move may not be adjacent. So a Surface plan is a set of moves that relocate individual subunits from one location on the perimeter to another, with the guarantee that a detailed sequence of consecutive Ghrist moves will exist that pass through the Surface plan way-points.

Figure 6.5 shows the generators where a unit can be added or removed from a Surface configuration to generate another valid Surface configuration. If local states at the single trace location, tr, satisfy $\hat{U}_0(tr) = EMPTY$ and $\hat{U}_1(tr) = OCCUPIED$ then we say the generator is an ADD, otherwise it is a $REMOVE$. A move for the Surface model is a $REMOVE$ followed by an ADD elsewhere. Technically, in Ghrist parlance, the catalog for the Surface model is the (infinite) union of all possible relative arrangements of a $REMOVE$ and an ADD whose labeled local states agree.

Figure 6.5 was not generated by hand. All valid Surface configurations containing eight subunits were enumerated using the Hamiltonian path constraint description above. The relative local contexts of adjacent empty space was stored and annotated with a label describing whether a subunit could be added or not. This set of annotated local contexts was processed by the C4.5 algorithm [63] in the Weka [29]data mining library to produce a shallow decision tree. The decision tree had 100% accuracy at determining what necessary local context was present to add a subunit, and was optimized upon valid configurations only. As a side effect, the data mining tool identified that a subunit could not be added if it created the patterns shown in Figure 6.6. The details are describe in appendix 2.

Lemma 1. *For any given surface adhering configuration, an additional module can move around the perimeter in a complete loop using the Ghrist motion catalog.*

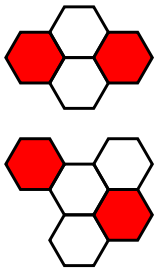


FIGURE 6.6. A Surface configuration never contains the above patterns of empty space (white) and robotic subunits (red).

Units, n	fails	trials	95% C.I. of P(fail)	
250	264	10000	.0233	.0297
500	7	10000	.0003	.0014

TABLE 2. Probability of *StoS* failing to find an improvement in random tasks decreases as the number of units in the random configurations increase.

Proof. By construction, discussed above. □

The Surface-to-Surface planning algorithm finds an admissible sequence of Surface moves in order to change one Surface adhering configuration into another. From Lemma 1 the resulting plan can be executed by a HMR constrained by the Ghrist, Claytronics or Chirikjian motion constraints. Each single move in the Surface HMR, however, is a concatenation of several single moves by the other catalogs (a so called, long-move) on account of the start and end location decoupling. The high level algorithm is outlined in (Algorithm 7). For clarity, the version presented here does not include a number of optimizations (see [45]). So this specific algorithm does not run in near linear time. However, the salient features relevant for the present discussion have been preserved.

The algorithm's main loop incrementally changes the current configuration (which is initially the start configuration) toward the goal configuration by applying valid surface moves determined by *improve*. The algorithm tracks a set P which represents placed subunits. Once a unit is placed, it is no longer considered by the *improve* sub-routine to be a possible subunit that can be moved. P is updated incrementally by *updateP* (Algorithm 8); a subunit is considered placed if it is adjacent to an already placed unit, and the goal contains a unit at the same location. It has been empirically determined that, on average, only $O(\sqrt{n})$ long-moves are required to transform a start configuration into the goal configuration [45].

The sub-routine *improve* finds a valid surface move that moves a subunit not in P to a location that would lead to an addition to P. The *improve* sub-routine is highly optimized elsewhere to maximize the chances that only a few moves are considered [45], but these optimizations are not included here. There is a small chance that no move will improve P in which case the planner fails to find a solution for the reconfiguration task at hand. Empirically this does not seem to happen often. In fact, the probability of failure tends toward zero as n tends to infinity (Table 2).

Algorithm 7 The 𝕊urface-to-𝕊urface finds a set of admissible moves to change a configuration c into a configuration c_{goal}. A set of placed subunits, P, is incrementally grown by calls to *improve*. *improve* searches for admissible moves to improve P.

$improve : \mathbb{S} \times \mathbb{S} \times \mathcal{P}(\mathbb{P}) \mapsto M$
$improve(c,\, c_{goal},\, P) \triangleq$
  for$(\forall s.s \notin P \wedge s \in c, \forall e.e \notin P \wedge e \in c_{goal}\ )$
    if $(match(c,\, s,\, REMOVE)))$
      if $(match(c - \{s\},\, e,\, ADD))$
        return (s, e)
throw error
$StoS$: $\mathbb{S} \times \mathbb{S} \mapsto M^k$
$StoS(c, c_{goal}) \triangleq M^k$
$P \leftarrow updateP(\emptyset, ORIGIN, c, c_{goal})$
while$(|P| < |c|)$
  $m \leftarrow improve(c, P, c_{goal})$
  $c \leftarrow c \cup \{m_2\} - \{m_1\}$
  $P \leftarrow updateP(P, m_2, c, c_{goal})$
  $append(m, M)$

The result of the Surface-to-Surface planner is a sequence of long-moves, representing location-to-location traversals round the perimeter of the intermediate configurations. Unwrapping the long-moves into a sequence of short-moves, compatible with other HMR models, can be done in near linear time [45]. This is possible because, on average, the perimeter distance for each long-move scales as $O(\sqrt{n})$, and the number of long-moves required to reconfigure also scales as $O(\sqrt{n})$. Thus it appears, empirically justified, that the average asymptotic performance of the Surface-to-Surface planner is nearly linear (subject to how close to constant time *improve* can be implemented) with an insignificant failure rate for large n.

We wish to understand several things about this algorithm. Why does the algorithm asymptotically fail less as $n \to \infty$, even though it is essentially a local heuristic? Why can the task be solved incrementally by growing a placed set, P ? Also, why can't the Ghrist HMR reconfiguration tasks be solved in a similar fashion *i.e.* what makes this particular HMR motion catalog *special*?

6.5. **The Surface Space is Highly Connected.** In general terms, a planning algorithm's task is to find paths through some space, C_{free}, for multiple start and end points. The difficulty of the task, and therefore the minimal complexity of a planning algorithm, is inherently coupled to the properties of the configuration space. Typically, for computational reasons, one approximates a continuous or otherwise complex, C_{free}, by discretization or sampling-based methods to yield a graph whose vertices are states in C_{free}. For HMR planning, C_{free} is naturally discrete; vertices of the space represent configurations, and edges represent admissible moves (or sets of moves in multi-move planning, but not considered here).

Algorithm 8 The set of placed units, P, is updated recursively. If a location, x, is in the current and goal configuration but not in P it is placed in P and $updateP$ is called on its neighbors.

$updateP : \mathcal{P}(\mathbb{P}) \times \mathbb{P} \times \mathbb{S} \times \mathbb{S} \mapsto \mathcal{P}(\mathbb{P})$
$updateP(P_{prev}, x, c, c_{goal}) \triangleq P$
$P \leftarrow P_{prev}$
  if$(x \in c \wedge x \in c_{goal} \wedge x \notin P)$
    $P \leftarrow P \cup \{x\}$
    for$(\forall q.isAdj(q, x))$
      $P \leftarrow updateP(P,\ q,\ c, c_{goal})$

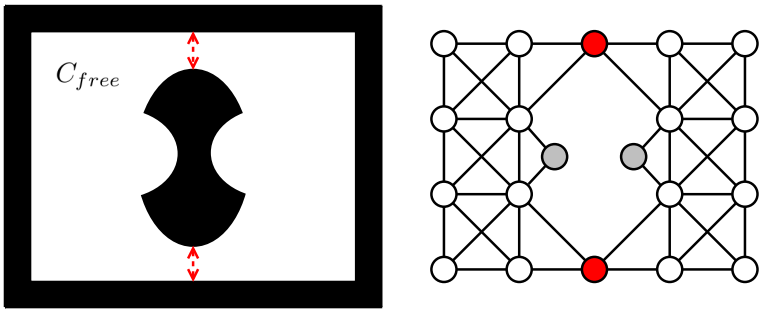


FIGURE 6.7. Left, a simple example of a planning space. The obstacle in the center causes bottlenecks in C_{free}(shown in red). Right, a graph approximation of the same space.

Bottlenecks in C_{free} are well known complications for planners[48]. A bottleneck is a suitably narrow subset within C_{free} that constrains different possible solution paths to go through it in order to traverse between much larger subsets of the space (Figure 6.7, red). Iterative or sampling-based planning algorithms that must 'discover' such bottlenecks computationally can face serious challenges as they may be naively expending precious computational time exploring irrelevant areas of C_{free} (Figure 6.7, gray).

Well founded graph-theoretic measures can concisely express what we mean by a bottleneck. Let $G = (V, E)$ be a simple unweighted graph. A cut, $W \subset V$, separates the graph into two sets of vertices, W and $V - W$. Let $deg(x \in V)$ be the degree of a vertex. Then the volume of a set of vertices $W \subset V$ is defined as $vol(W) \triangleq \sum_{i \in W} deg(i)$. The cost of a cut on the graph is $cut(W) = |\{x = (u, v) | x \in E \wedge u \in W \wedge v \notin W\}|$, in other words, the number of edges crossing the cut.

The Cheeger constant of a graph, h_G, is a measure of "bottleneckedness" which finds a small cut that separates the graph into two large volumes [16]. It is defined as:

$$h_G = \min_S \frac{cut(S)}{min\{vol(S), vol(V - S)\}}$$

Directly measuring the Cheeger constant for a graph is computationally intractable. It is, however, bounded by the 2^{nd} smallest eigenvalue of the Laplacian matrix (which is also known as the algebraic connectivity of the graph [16]).

The Laplacian L matrix of a graph is:

$$l_{i,j} := \begin{cases} 1 & if\ i = j\ and\ deg(v_i) \neq 0 \\ -\frac{1}{\sqrt{deg(v_i)deg(v_j)}} & if\ i \neq j\ and\ v_i\ is\ adjacent\ to\ v_j \\ 0 & otherwise \end{cases}$$

If L has the eigenvalues $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$ then the Cheeger constant h_G is bounded by:

$$\sqrt{2\lambda_2} > h_G \geq \frac{\lambda_2}{2}$$

λ_1 is 0 for all Laplacians [16]. λ_2 is known as the algebraic connectivity of the graph. If a graph has a low Cheeger constant, this implies there are small cuts that can separate large volumes of the graph. This captures the essence of bottlenecks; paths between the two volumes must be routed through a small corridor.

Our hypothesis is that the configuration space of the Surface model has fewer bottlenecks compared to the Ghrist model of the HMR. We will use algebraic connectivity and its relation to the Cheeger constant to measure the severity of bottlenecks in C_{free} for the Ghrist model and the Surface model. However, first note that the atomic moves in the Ghrist motion model represent a single subunit moving an adjacent location, whereas Surface moves represent a single subunit moving a number of lattice locations. To compare the models fairly, we created an analogous definition of a long-move for the Ghrist model to be a sequence of consecutive admissible moves applied to a single subunit *i.e.* all the locations that a single subunit can reach while the other subunits remain fixed. In addition, because both model' catalogs don't permit gross topological changes in the morphology, we only study configurations that do not contain enclosed space.

For the configuration graphs containing up to 7 subunits, it is possible to construct the Laplacian and calculate the algebraic connectivity directly. The results are shown in Table 3. However, the reconfiguration graphs differ very little with low numbers of subunits for the two models. With few subunits, there are not enough permutations of possible local contexts to differentiate the models. The difference between models only becomes apparent at higher complexity levels.

Unfortunately, expanding the configuration graphs containing larger number of subunits quickly becomes intractable. So in order to estimate the properties of the Cheeger constant at higher complexity levels, *i.e.* subunit numbers, we use a sampling methodology. First, we generated a random Surface adhering configuration containing n units by iteratively uniformly selecting an ADD action to a growing configuration (starting from the ORIGIN). Second, we applied 1000 random moves from the respective motion catalog (long-moves for the Ghrist model), so that the initial Surface configuration diffuses into a model-specific area of the configuration space. Finally, the model-specific configuration reached is used as a starting location for taking a sample. Examples of configurations generated by this procedure are shown in Figure 6.4.

	Ghrist			Surface		
n	$\|V\|$	$\|E\|$	λ_2	$\|V\|$	$\|E\|$	λ_2
2	6	15	1.2000	6	15	1.2000
3	33	168	1.1429	33	168	1.1429
4	176	1431	1.1186	176	1431	1.1186
5	930	10836	1.1033	900	10332	1.1111
6	4878	75945	1.1	4482	67725	1.0978
7	25480	506394	1.0872	21910	417042	1.0909

TABLE 3. The number of vertices, edges and algebraic connectivity (λ_2) of the Ghrist and Surface model' reconfiguration graphs generated by different numbers of subunits (n). The states spaces are identical up to $n = 4$, and only differ marginally at $n = 7$

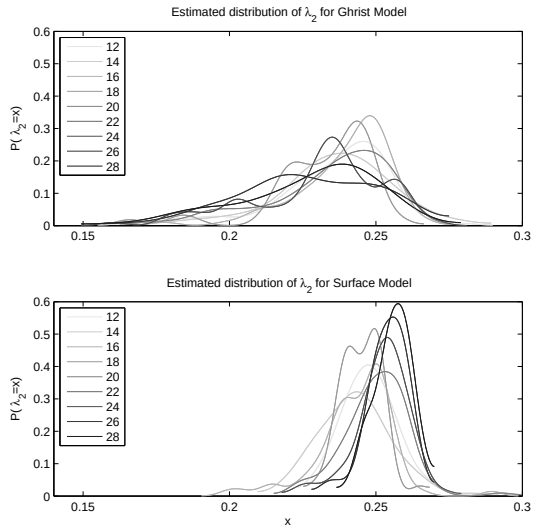


FIGURE 6.8. The estimated densities of λ_2 of the Laplacian after sampling 100 sub-graphs from the reconfiguration spaces of the Ghrist model and the Surface model.

A sample sub-graph is generated by performing a breadth first search to a depth of two from the sample location (the sample location, its neighbors, and its neighbor's neighbors). The algebraic connectivity may be computed for this sub-graph, and should correlate to the global algebraic connectivity. This procedure was applied to 100 samples for each complexity level under study. The smoothed results are shown in 6.8.

For the Ghrist model, Figure 6.8 shows that as the number of units in the configuration increases, so the spread of λ_2 increases, and the mean diminishes. This suggests that our sampled sub-graphs are increasingly likely to contain bottlenecks. For higher numbers of subunits this suggests that

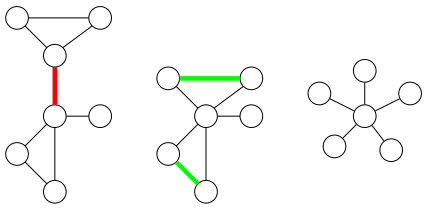


FIGURE 6.9. Graph minor operations, graphs to the right are minors of those to the left. Red denotes an edge contraction operation, and green an edge deletion.

the Cheeger constant is tending toward 0. For the Surface model the reverse seems to be true. The spread of λ_2 is decreasing, and the mean increasing. Bottlenecks seem to be sparser as we add more units to the surface configurations.

Our interpretation for the Surface model is that when there are very large numbers of subunits, the movement of a particular subunit is relatively unrestricted; it is able to move anywhere on the surface. Interaction between subunits mainly occurs at the local level, whose importance diminishes as the number of units grows. Thus local interactions that cause bottlenecks become less likely, and the mobility of subunits on the perimeter increases. For the Ghrist model, it only takes two kinks on the surface to divide the perimeter into two classes that units cannot move between (see the H configuration in Figure 6.17 in next section). As the number of units grow, so the probability of two or more kinks being found somewhere on the perimeter tends toward certainty.

The implication of Figure 6.8 is that the Surface model has fewer bottlenecks compared to the Ghrist model. The sparsity of bottlenecks in the Surface model explains why a greedy planning procedure, such as the one employed in the Surface-to-Surface planner, suffices in an increasing proportion of cases as n grows.

6.6. **Graph Minor Sub-Structure.** The previous section uses Spectral Graph theory in order to explain when greedy planning methods suffice in a reconfiguration state space. Within this next section we introduce the use of the Graph Minor theory applied to the analysis of SRS state spaces, firstly as a compact, precise notation for representing that one state space is a constrained version of another, and secondly, as a tool that reveals startling differences between the easy and hard planning spaces as subunits are added. This last point in particular partially explains why efficient planning methods may only exist for some planning state spaces.

Definition 2. A graph, H, is said to be a *minor* of a graph, G, denoted $H \leq G$ if there exists a sequence of edge deletions, contractions and vertex deletions to change G into H. [18]

Figure 6.9 illustrates the basic graph minor modification operations. The graph minor relation is a compact notation for describing when one state space can be executed upon another model; for describing when one SRS motion model is a constrained version of another. Consider the similar sub-graph relation:

Definition 3. A graph, H, is said to be a *sub-graph* of a graph, G, if there exists a sequence of edge deletions and vertex deletions to change G into H. [18]

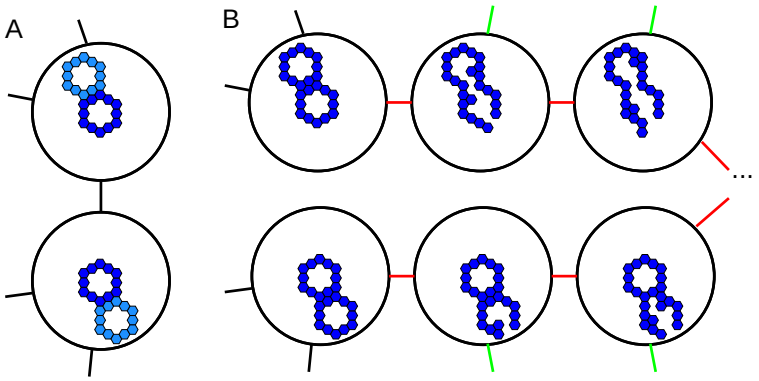


FIGURE 6.10. A. a meta-module can tunnel through another, represented by a single edge in the planning state space. This single move in the meta-module state space is implemented using a sequence of moves from the underlying state space. The transient moves used in the Claytronics state space have used the edge contraction operation to form atomic moves in the meta-module state space (B, red). The edge contractions, plus pruning of non-conforming meta-module states and moves (green) show that the meta-modularization is a graph minor of the Claytronics state space.

The sub-graph relation lacks edge contractions. Now consider the meta-module reconfiguration state space graph containing k meta-modules, $\mathbb{M}_k$, and the Claytronics state space which contains 12 subunits for every meta-module, $\mathbb{C}_{12k}$ (Figure 6.10). Meta-module movements are built from sequences of underlying motion primitives, so although each $\mathbb{M}_k$ vertex is present in the $\mathbb{C}_{12k}$ graph, each edge of the $\mathbb{M}_k$ graph represents a *sequence* of underlying $\mathbb{C}_{12k}$ edges. Thus $\mathbb{M}_k$ is not a sub-graph of $\mathbb{C}_{12k}$, yet it is a graph minor, *i.e.*, $\mathbb{M}_k \leq \mathbb{C}_{12k}$. Similarly, we can summarize all the discussed HMR state spaces using graph minor nomenclature as $\mathbb{J}_k \leq \mathbb{C}_k \leq \mathbb{G}_k \leq \mathbb{S}_k$, where $\mathbb{J}_k$, $\mathbb{C}_k$, $\mathbb{G}_k$ and $\mathbb{S}_k$ stand for the Chirikjian, Claytronics, Ghrist and Surface model reconfiguration state spaces containing k subunits respectively.

The minor relation is a compact, precise notation for expressing what we mean by one state space is a constrained version of another. Butler *et al.* [7] present an argument that their cubic SRS model is generic because it can be instantiated by various existing SRS motion catalogs. If we denote their cubic SRS reconfiguration state space containing k subunits as $\mathbb{B}_k$, the meta-modularized M-Tran state space as $\mathbb{T}_k$, we can rewrite one of the instantiations described in the work as $\mathbb{B}_k \leq \mathbb{T}_{4k}$, as four M-Tran units were required to cooperate in order to achieve the minimum motion requirements of their model.

While the graph minor relation is a useful notation for describing relationships between different SRS motion catalogs, we now look at the family of state space graphs of an individual SRS catalog generated by different numbers of subunits, *e.g.*, $\mathbb{S}_n$, for $n > 1$; to understand how the planning problem changes as more subunits are added. Intuitively one might presume that the state space graph for a HMR model containing i subunits will share similarities with the state space containing

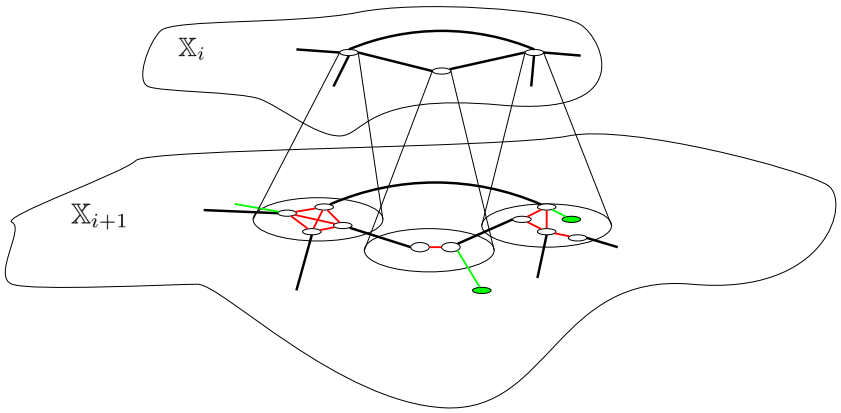


FIGURE 6.11. To globally show $\mathbb{X}_i \leq \mathbb{X}_{i+1}$ we can define a local relationship between the graphs, and show that this relationship locally adheres to the minor relationship. Red denotes edge contractions, and green, edge and vertex deletions. The global minor can be proved by stitching together the local minors.

$i+1$ subunits. We address this question formally and find a significant difference between the state spaces generated by models that are hard to plan for, *e.g.*, the Ghrist catalog, and models that have efficient solutions in existence, *e.g.*, the Surface catalog or a meta-modularized state space.

For the Surface model, the i reconfiguration graph is a graph minor of the $i+1$ reconfiguration graph, $\mathbb{S}_1 \leq \mathbb{S}_2 \leq \ldots$. This does not appear to be true of the configuration graphs generated by the Ghrist motion catalog. In fact, the counter-examples for the Ghrist case are caused by the very cases where bottlenecks are found. Similar to the $\mathbb{S}$ case, the HMR meta-modularization example is also well-ordered by the minor relation, $\mathbb{M}_1 \leq \mathbb{M}_2 \leq \ldots$. As will be discussed further later, graph minor ordering in the reconfiguration state spaces has significant implications for the motion planning problem, and is likely to be the mechanism for explaining why one motion model admits efficient planning and others do not.

To show that a reconfiguration state space, $\mathbb{X}_i$ is a minor of $\mathbb{X}_{i+1}$, we utilize the fact that each vertex of the reconfiguration graphs is labeled by the arrangement of subunits on a common embedding lattice. This labeling scheme permits a vertex, v_i of the $\mathbb{X}_i$ graph to be associated with a group of vertices, $\bar{\mathbf{v}}_{i+1}$, in the $\mathbb{X}_{i+1}$ graph that corresponds to possible locations a subunit can be added to the v_i configuration to generate a configuration within $\mathbb{X}_{i+1}$. This observation implies that a local area of the $\mathbb{X}_i$ graph has a corresponding local area in the $\mathbb{X}_{i+1}$ graph.

To show globally that the $\mathbb{X}_i$ graph is a minor of the $\mathbb{X}_{i+1}$ graph, it is sufficient to prove that: for every vertex, v_i, in the $\mathbb{X}_i$, that v_i's local graph neighborhood is a minor of $\bar{\mathbf{v}}_{i+1}$ graph neighborhood, and that these local neighborhoods are connected in the same topology. This is summarized diagrammatically in Figure 6.11.

The sketch of the proof to show that $\mathbb{S}_i \leq \mathbb{S}_{i+1}$ is as follows. Any configuration adhering to the Surface model implies that if a module can move at all, then it is free to move in a complete loop around the exterior. The S_{i+1} space contains one extra subunit. We show that the subunit can always move out of the way, in order to let any move that existed in the S_i graph take place.

Prevention of a move in the $\mathbb{S}_i$ state space for the $\mathbb{S}_{i+1}$ state space can only occur if the extra subunit in the $\mathbb{S}_{i+1}$ state space interferes with the local support that determined the move's admissibility. We argue geometrically that for large configurations, there always exists a potential location for the added subunit that lies outside of the local support locations of the $\mathbb{S}_i$ move. The finite size of the motion catalog's local supports, plus the total mobility of the additional subunit, implies that "get out of the way" moves always exist in the $\mathbb{S}_{i+1}$ state space. The "get out of the way" move edges can be contracted to generate the $\mathbb{S}_i$ graph, thus showing that $\mathbb{S}_i \leq S_{i+1}$. This argument does not follow for the Ghrist model because, in general, the extra subunit does not always have enough freedom to "get out of the way" of the local supports that determined the admissibility of a move.

%CHANGE3A%

The geometric proof is constructed such that the two local supports of finite size cannot interfere with the existence of an additional subunit placed somewhere once the configuration is above a certain size (a big configuration always has room for a subunit to be added away from the locality of a move we are trying to avoid). It is difficult to argue this in the 2D case, so we consider the configuration projected onto a line along its longest axis. Now the size of the support is bounded by an interval, and if a configuration is large enough (longest axis scaling at $\sqrt{n}$), there is always room elsewhere on the line for a module to be added, away from the bounds of **any** two supports. The benefit of this proof strategy is that it is human readable, but by grouping all the all the moves into one bounded size, and checking for interference only on a line, the result is that we only prove the graph minor ordering to hold for $n > 631$. We actually believe it to hold for $n > 0$, but this proof would require considerable enumeration of cases by machine and would be unreadable. This is the reason for choosing the proof strategy that we did.

Formally, we introduce the notions of local structure in a reconfiguration graph around some vertex, and an inherited local structure which represents the analogous locale in a reconfiguration graph generated by adding a unit.

Definition 4. The *local structure* for a configuration, v, is all configurations reachable by a single Surface move (remember a move is a REMOVE followed by an ADD from the Surface catalog, Figure 6.5).

Definition 5. The *inherited local structure* for a configuration, v, is all possible configurations generated by applying an ADD from the Surface catalog to v. (Figure 6.5)

Lemma 6. *Any two configurations belonging to a vertex's inherited structure have a valid move between them.*

Proof. This follows from Lemma 1. Thus the inherited local structure forms a clique of configurations connected by moves.

To show that the *inherited local structure* preserves analogous moves that existed in the *local structure*, we first show that an extra subunit can always be added at a location that is far enough

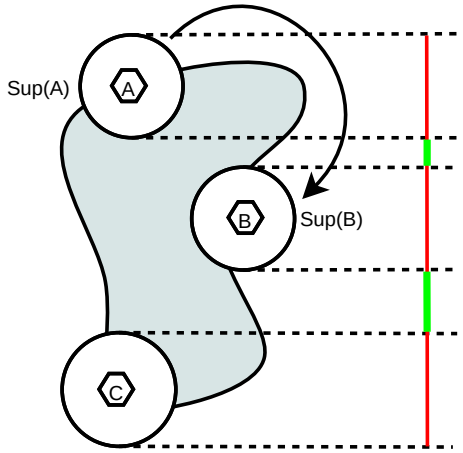


FIGURE 6.12. Whether or not a subunit exists at location C does not affect a move between A and B because its support does not intersect A or B's. Rather than showing this in 2D, we project the support areas onto a line parallel to the widest diameter of the shape. Showing the supports do not intersect is simplified to showing the projected support intervals do not overlap.

away from the start and end of the move so that it does not affect the local support that determined the moves admissibility (sketched in Figure 6.12). If a move is between position A to position B, we need to show that $\forall A \forall B \exists C.(sup(A) \cap sub(C)) \cup (sup(B) \cap sub(C)) = \emptyset$. There are a variety of ways to show this, for simplicity, in the following proof we project the support areas onto a line parallel to the widest diameter of the configuration. □

Lemma 7. *All generators of the Surface catalog have a width of less than 5.*

Proof. See Figure 6.5. □

Lemma 8. *A connected configuration containing 631 or more subunits has a large diameter of at least 29.*

Proof. The configuration with the smallest large diameter occurs when subunits are arranged into a perfect hexagon. 631[1] subunits can be arranged into a large hexagon of diameter 29. Moving any subunit, or adding more subunits, will only increase the large diameter. □

Lemma 9. *For a Surface configuration, within an columnar interval of width 7, a valid ADD location and its complete support is contained.*

Proof. First, a shape of width 7 is slid over empty space toward the configuration (Figure 6.13) until intersecting a subunit. The lower row of hexagons comprising of the shape will then contain between one and seven subunits and the remainder shall be empty (by construction). We will now consider different cases of how the bottom row can be occupied in order to show that regardless of how, there is always a location where an extra subunit can be added.

[1] A Hexagonal Number

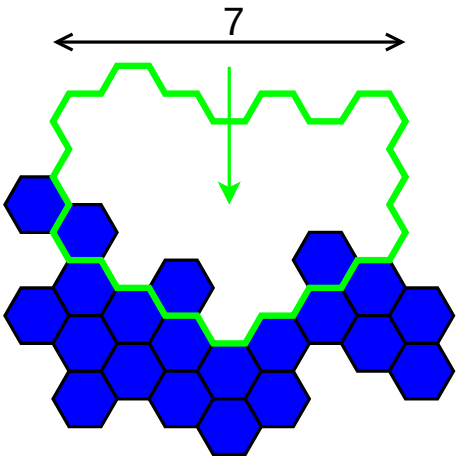


FIGURE 6.13. Lemma 9 is shown by sliding the shape shown in green (of width 7) toward the configuration until it overlaps one or more subunits on its lower edge.

A and B of Figure 6.14 reflect the cases of when the bottom row contains only one subunit. In each case a subunit can be added using Surface ADD E.1 (Figure 6.5). When two subunits are present and adjacent, Figure 6.14 C and its generalizations demonstrate Surface ADD E.2 can be used to add a subunit. When the two subunits are a distance of one from each other, case D is relevant. Case D can only occur if additional subunits are found adjacent to the empty location (light blue), because otherwise the configuration would be invalid (Figure 6.6). With the implied extra subunits included, Surface ADD E.4 (Figure 6.5) is applicable. Another possibility when a pair of lower row subunits are at a distance of one is case E, this however, is an impossible Surface configuration (Figure 6.6), but regardless, an applicable ADD location exists. When two subunits are at a distance greater than two, such as in the case F, it is clear one subunit no longer becomes relevant to determining an ADD applicability. For cases with more subunits, the above arguments are trivial to extend (Surface ADD E.3 is used when case B is extended to three subunits). Therefore, within an interval of 7, a valid ADD location can always be found, regardless of the specifics of the Surface configuration. □

Lemma 10. *On a line of length 29 or greater, if two intervals of width 5 are present, then an interval of width 7 can be found which intersects neither.*

Proof. The worst placement of the intervals of width five are shown in Figure 6.15 for a line of width 28. Clearly extending the length of line by one will permit space for an interval of width 7 to be inserted without overlap. □

Lemma 11. *For any move between location A to location B on a Surface configuration containing 631 subunits or greater, there exists a location C where a subunit can be added, whereby the support of A and B do not intersect the support of C, i.e.,* $\forall A \forall B \exists C.(sup(C) \cap sup(A)) \cup (sup(C) \cap sup(B)) = \emptyset$.

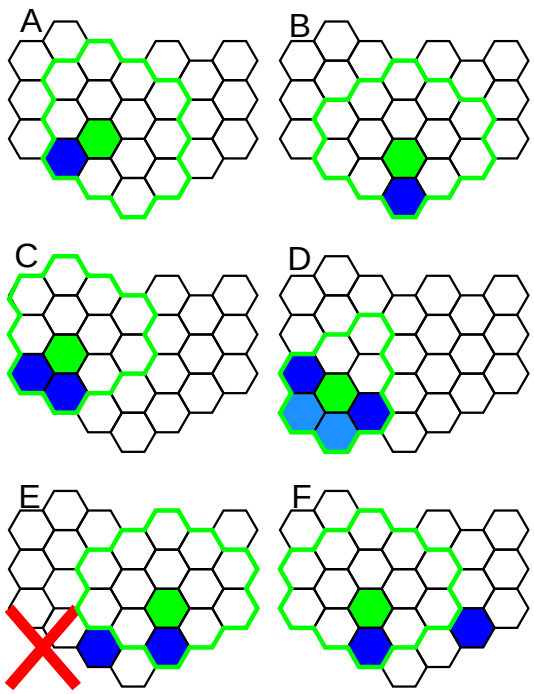


FIGURE 6.14. The major cases for consideration of how the shape in Figure 6.13 can be occupied with subunits. The area marked with a green perimeter labels the location of an applicable support for some Surface ADD generator. The location of where the subunit can be added is shown in green.

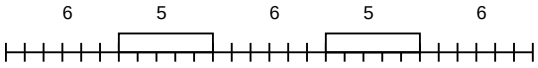


FIGURE 6.15. On a line of length 28, two intervals of width 5 can be placed such that an addition interval of 7 cannot be placed without intersecting one of them.

Proof. Projecting the supports of A and B onto a line parallel to the line defining the large diameter of the 631 subunit sized configuration yields two intervals of size 5 (by Lemma 7) on a line of length 29 (Lemma 8). An interval of width 7 shall exist on this line that does not intersect either of the intervals of size 5 (Lemma 10). Somewhere within the area of the configuration that would project to the interval of size 7 exists and ADD location C (Lemma 9) which cannot intersect the supports of A or B. □

Remark. This also holds for configurations of any size, but the proof is cumbersome and involves extensive enumeration of cases.

Lemma 12. *For every move $A \to B$ in a local structure between configurations v and u, there exists at least one pair of configurations v' and u' in the inherited structure between the same locations.*

Proof. A configuration can be represented as a set of subunit locations. Let $v = X \cup \{A\}$ and $u = X \cup \{B\}$. We simply need to find an ADD location C that can be added to v and u such that it

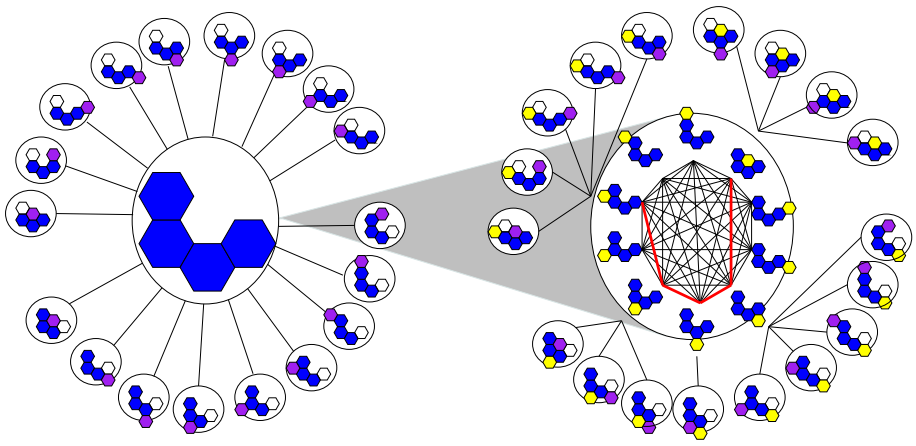


FIGURE 6.16. A local structure (left) is a minor of the inherited local structure (right). The left central vertex is surrounded by all configurations reachable by a move (its local structure). The right central vertex contains the inherited structure for that vertex (a clique), yellow denoting where an additional subunit has been added. For every move in the local structure (white to purple), a comparable move can be found in the inherited structure with an addition subunit added, denoted by the vertices joining the central vertex. The red lines within the inherited structure shows which moves are required to move the additional subunit around to "get out of the way" so that all analogous moves can execute. Deleting all black edges in the inherited clique followed by contracting the red edges reproduces the local structure.

does not interfere with the support of A and B that enabled the move $A \to B$ to take place. Lemma 11 shows such a C exists when there are more than 631 subunits in the configuration. Thus a C always exists such that $v' = X \cup \{A, C\}$, $u' = X \cup \{B, C\}$ and the move $A \to B$ is still valid. □

Lemma 13. *The local structure of a vertex, v, is a graph minor of its inherited local structure v'.*

Proof. Each neighbor, u_i, in the local structure of v, represents a valid move between the configurations v and u_i. By Lemma 12 $\forall u_i$, there exists a location z_i that permits a move between $v \cup \{z_i\}$ and $u_i \cup \{z_i\}$. By definition, the configuration $v \cup \{z_i\}$ is in the *inherited structure*. By Lemma 6 there exists a move between all $v \cup \{z_x\}$ configurations. If all moves between $v \cup \{z_x\}$ are contracted and all edges not $v \cup \{z_i\} \to u_i \cup \{z_i\}$ in the inherited structure are deleted, the remaining edges are the local structure (see Figure 6.16)

□

Theorem 14. *The state space of the Surface model containing i subunits is a graph minor of the reconfiguration space containing $i + 1$ units, $\mathbb{S}_i \leq \mathbb{S}_{i+1}$ for $i \geq 631$.*

Proof. By Lemma 13 every vertex in the i graph is a minor of the inherited graph. For a pair of configurations in the i graph, $u = X \cup \{x\}$, $v = X \cup \{y\}$ with a move between them, $x \to y$, Lemma 13 states an ADD location on each, $z_u$and z_v exists such that the same move can take place in the

inherited structure, $X \cup \{x, z_u\} \rightarrow X \cup \{y, z_u\}$ and $X \cup \{y, z_v\} \rightarrow X \cup \{x, z_v\}$. By Lemma 6, a connecting move between the local minors exists between $X \cup \{x, z_v\}$ and $X \cup \{x, z_u\}$ and thus we can compose all the local minors of Lemma 13 into a graph and edge contracting the connecting moves to produce the reconfiguration graph containing i subunits. □

Remark 15. For an alternate viewpoint on the same result, we could entirely skip the composition of local graphs. An extra subunit can be added, and moved out of the way in order to realize all sequences of realizable moves (Lemma 12). However, this loses sight that there is a notion of locality relating the local structure to the inherited local structure through the embedding space. This becomes important when we consider the counter example for the Ghrist model.

Conjecture 16. *The Ghrist reconfiguration graph containing i units is not a graph minor of $i+1$ for i greater than some constant.*

Our above construction of graph minor for the Surface model does not hold for the Ghrist configuration graph because an additional subunit in the inherited local structures does not, in general, form a clique structure. Thus, while a location may exist for every local move that permits the move to take place in the inherited structure, there may not be connections between these locations. Figure 6.17 shows a counterexample where the inherited structure is disconnected. In these cases the local structures are not minors of the inherited structures, and so a global minor cannot be constructed from a composition of local minors.

Interestingly, if the H configuration counterexample in Figure 6.17 is used as a starting point for a sub-graph sample for the procedure in section 6.5, then the resulting sub-graph yields an λ_2 of just 0.03. This classifies the configuration as the most bottlenecked configuration encountered. It appears that the areas of the reconfiguration space where the graph minor relation breaks down is also where bottlenecks appear.

%CHANGE3B%

Next we consider meta-modularization. Meta-modularization was first formulated on a cubic lattice [65] with a tunneling procedure that allowed a meta-module to move from anywhere on the surface, to anywhere else on the surface of the connected component (figure 6.18). Changes in gross topology were not excluded, unlike the Ghrist and Surface model, so that an additional constraint that the aggregate remains connected must be enforced explicitly.

Meta-modularization has efficient planning algorithms developed upon it, and it is also ordered by the minor relation, $\mathbb{M}_1 \leq \mathbb{M}_2 \leq \ldots$. Reusing the above logic for the Surface model ordering proof, above a certain size there is always enough exposed surface upon which a subunit can be added that will not disrupt any previous moves. The only new consideration is that the tunneling procedure can only move units around on the same connected component, but as there is an explicit constraint enforcing that the sub-units remain a single connected component, this taken care of.

Theorem 17. *For the meta-module state space, $\mathbb{M}_1 \leq \mathbb{M}_2 \leq \ldots$.*

Proof. argued above. □

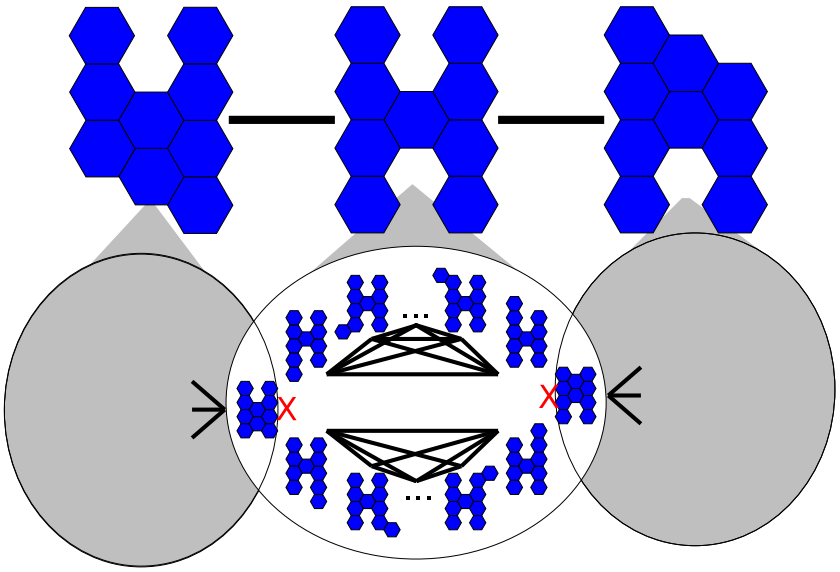


FIGURE 6.17. A counterexample case for the Ghrist model. Two neighbors in the local structure of the central H configuration are shown (top). The induced local structure of the H configuration is divided into four connected components, two cliques and two unconnected vertices. The local structure cannot be reconstructed from edge deletions and contractions of the inherited structure, and thus is not a graph minor.

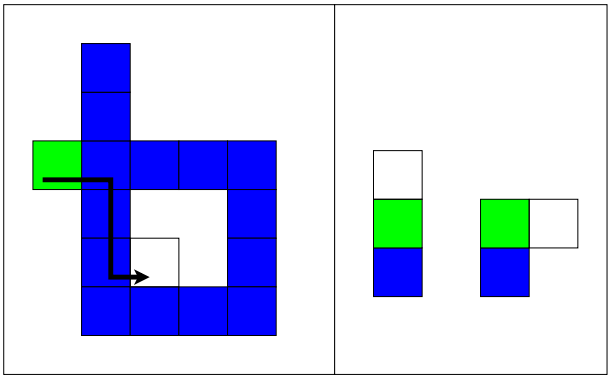


FIGURE 6.18. In meta-modules $\mathbb{M}$ state space, units can tunnel from the surface to another surface location. The supports of the start and the end of the move are thus very basic (right), though note that the catalog is infinite in size as it encompasses every possible start/end location and orientation.

What is perhaps more illuminating is SRSs that are not well-ordered by the minor relationship. Consider a modified meta-module system, $\mathbb{M}^S$ whose subunits can only move to perimeter locations on the same perimeter (by moving around the surface). In this case, the move shown in figure 6.18 would not be able to take place, because it switches from the external surface to an internal surface. The modified meta-module' state spaces we believe are not ordered by the minor relationship.

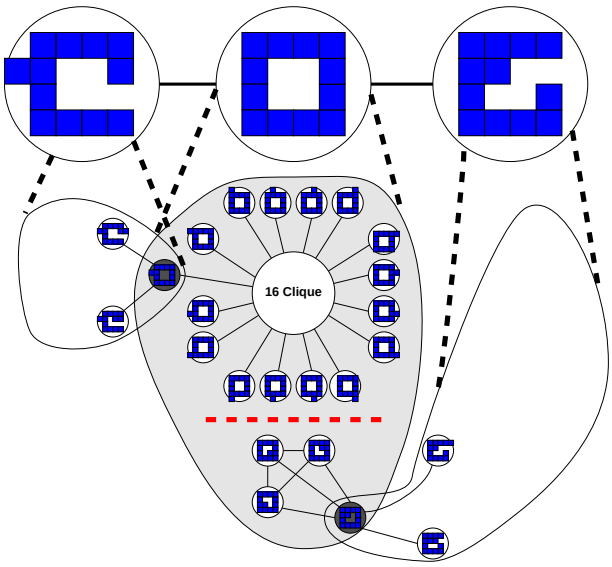


FIGURE 6.19. In the inherited local structure for a hollow square, there are two major classes of where a subunit can be added. Inside and outside. All outside configurations have a valid move between them (forming a 16-clique). Similarly, the inside inherited configurations form a 4-clique in the inherited local structure. Importantly, there are no valid moves between the inside and outside inherited configurations, so analogous routes in the local structure, do not occur in the inherited local structure.

The breakdown of the minor relationship occurs locally for moves where new surfaces are created. An example is shown in figure 6.19. New surfaces in the configuration split subunits into groups of mobility. If a subunit is removed from a configuration containing holes, there is the potential for many more moves to exist in the similar configuration containing one less subunit. Thus, locally, edges would have to be added, and **not subtracted or contracted** to create an equivalent graph structure. So we can therefore conclude that the surround near hole containing configurations cannot be a minor. That said, this argument only hold locally, so we can only conjecture that $\mathbb{M}_i^S \not\leq \mathbb{M}_{i+1}^S$ as it is difficult to exclude the possibility there maybe some other way of finding the $\mathbb{M}_i^S$ structure from the $\mathbb{M}_{i+1}^S$ graphs using edge deletions and contractions.

Decoupling the start and end positions of moves is the primary reason why minor ordering is found in the reconfiguration graphs of the easy planning spaces studied here. It must be noted though, that the minor ordering is a global structural property of the reconfiguration graphs, and not a consequence of the representation used to describe the motion catalogs.

Graph Minor Theory is a powerful, modern tool. Many properties are persevered or bounded by taking minors. If a graph $H \leq G$ and G can be drawn in some topological space without edge crossings (*e.g.* a planar graph, or a generalization thereof) then H can be too. H is no more complex (in a topological sense) than G.

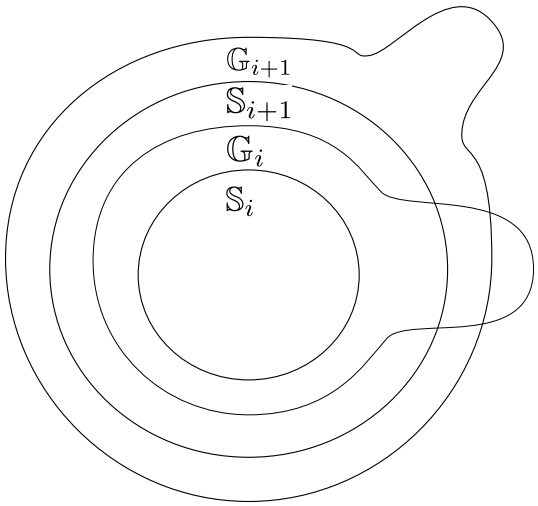


FIGURE 6.20. Each Surface state space can be nested within the next. While a Surface state space is always found within a Ghrist state space, each Ghrist state space contains an area that is not contained with its child. It is for this reason that Ghrist problems cannot be solved efficiently over the entirety of the state space.

It is somewhat difficult initially, due to the simplicity of the graph minor relationship, to see the consequences of minor ordering. Recall the easier-to-grasp concept introduced first that if $H_k \leq G_k$, and H and G are distinct SRS motion models, then plans for the hardware of H can be run on the hardware of G *e.g.* H could be a meta-modularization of G. Generalizing this, we can now see that if $H_i \leq H_{i+1}$ that plans for the H_i reconfiguration graph can be instantiated on the same hardware, differing only in that an extra subunit has been added (H_{i+1}). Thus, plans in H_i can be reused, and augmented, to form plans in H_{i+1}, so planning in these well-ordered spaces can be achieved in an incremental, local and recursive manner.

In contrast, the hard planning spaces, like the Ghrist model, do not permit this efficient strategy. As $\mathbb{G}_i$ is not a minor of $\mathbb{G}_{i+1}$ this implies that both edge deletions and additions must be used to modify $\mathbb{G}_{i+1}$ into $\mathbb{G}_i$ (and in fact, vice versa). So, a plan that worked for a $\mathbb{G}_i$ planning task may not always operate in an analogous manner for the $\mathbb{G}_{i+1}$ case, because it may of utilized an edge in the reconfiguration graph that no longer exists.

Like the subset relation, the graph minor relation induces a partial order on a set of elements. Partial orders can be summarized graphically using a nested set notation. The observations in this section about how the Ghrist state spaces relate to the Surface state spaces and between themselves using the graph minor relation are summarized in Figure 6.20.

6.7. **Discussion.** Self-reconfiguring systems are a desirable future robotic technology. Unfortunately, practical implementations of SRS tend to have awkward motion constraints that make planning computationally difficult. To get the full benefits of SRSs we require efficient motion

planning algorithms so that SRS deployments can reconfigure on demand in response to environmental challenges.

So far efficient motion planning algorithms have been developed on a somewhat *ad hoc* basis, wherein researchers have looked carefully at each instantiation of SRS architectures and carefully chosen motion catalog restrictions. So far, we have lacked a theoretical understanding of why some classes of SRS are good to plan within and some are not. Our work is an attempt to elucidate the structure of SRS reconfiguration spaces, which could be exploited in planning algorithms. We applied graph-theoretic techniques to sample the reconfiguration space in order to quantify the presence of bottlenecks, and we identified a graph property that separated an easy to plan with SRS model from a harder one. These are general methodologies with computational implications for a much larger class of SRS.

Meta-modularization has been a common tactic in the SRS community for isolating troublesome motion constraints within an abstraction. Meta-modularization often involves the definition of a tunneling procedure that allows a peripheral meta-module to appear anywhere else on the perimeter of the configuration. The unconstrained movement of meta-modules around the perimeter using a tunneling procedure is similar to the Surface model's long-move motion primitives (though the Surface model does not permit movements to locations on the perimeter that cause Surface violations). Both meta-modularization and the Surface model configuration graphs are well-ordered by the graph minor relationship, which we believe goes some way towards explaining why these approaches make planning easier. It is clear that the Surface model's motion catalog is *far less restrictive* than the strategy offered by meta-modularization though. The configurations adhering to the Surface model's constraints occupy a much larger volume of HMR configurations, and thus sacrifice less generality in the configurations that can be planned with efficiently, than an alternative meta-modularization approach would.

The reason why Surface constraints are significantly less restrictive is because they are defined as addition *local* constraints describing where a subunit cannot stop along a motion *path*. There is no restriction that the local constraints to be defined in global terms (*e.g.* at specific points on a globally defined grid spacing as in meta-modularization). It seems entirely plausible that with a set of geometric path primitives (path segments and perhaps more generally branches), and with the insights of this paper (keeping algebraic connectivity high, and looking for graph minor ordering), that a set of *local* constraints that constrain an underlying model only a little, but simplify planning significantly, can be elucidated automatically.

Constraining a motion model only a little implies that only a small volume of the target general state space cannot be represented. In chapter 5, we utilized the Surface state space as an efficient basis for long range planning across the more general Claytronic motion catalog, with occasional recourse to more expensive but general search methods. Although the overall algorithm targeted the Claytronics motion catalog, by finding a large subspace that was efficient to work within, the size of the 'difficult' part of the remaining space was greatly reduced. Thus, overall the algorithm could achieve near linear performance, as demonstrated using a large number of randomly generated configurations, over a large proportion of the target state space.

6.8. **Conclusion.** SRSs need efficient motion planning algorithms, but developing them has been difficult because of the inherent high dimensionality and complexity (due to motion and shape constraints) of the problem. An efficient SRS motion planning algorithm must exploit local and global structure. In this work we have shown that even in cases where the basic state space of a planning problem may be complex, specific subspaces may admit much more interesting structure that can be gainfully utilized for planning. We have made precise what structure is required of the subspace, and moreover, we have shown how one can characterize this structure using general and powerful mathematical tools that are applicable to a large class of SRS problems. One promising direction for future work is to try to utilize these conceptual ideas to develop techniques that automatically discover efficient subspaces from more complex self-reconfiguration models.

7. Conclusion

%CHANGE2A%

Self-reconfiguring robotics systems offer possibility of a robotic systems that can change shape to suit the task at hand, or in response to environment challenges. A key piece of software in such an envisaged system is a planning algorithm that can coordinate the aggregate of cooperating subunits. The engineering of physical SRSs is still in its infancy, so we do not yet know what a field deployable SRS might look like yet. This uncertainty of the ideal physical realization has led to numerous different artificial models being suggested, each with its own particular flavor of motion constraints.

In addition to the uncertainty of how a practical SRS would move, we also do not know exactly what tasks such a system would undertake in the field. It is for these reasons in this thesis we have studied useful characterizations of state spaces with respect to planning for the shape-to-shape planning problem. While we have used the HMR extensively as a reference SRS, the characterizations of chapter 6.5 are applicable to any SRS whose state space can be represented by an undirected graph. This encompasses all lattice and unit-compressible SRSs, but also chain and hybrid SRS system if the subunits are restricted to lattice coordinates during reconfigurations.

The shape-to-shape planning task is not exclusively the only method for SRS control. It is, however, the most obvious method for utilization into a larger control system. With a robust shape-to-shape control methodology, high level tasks can performed by changing the SRS into different shapes in response to sensor information, using an as-yet-unknown shape trajectory planner. An alternative approach to SRS control that we have not not studied is using local self-organization to achieve high level tasks goals (such as locomotion). That said, while our focus was for the shape-to-shape planning, it is probable that the nice planning spaces characterized in chapter 6.5 will be useful substrates for self-organizing control too as a feature of these spaces were that subunits do not often get stuck.

In the shape-to-shape planning domain, there have been a number of different efficient algorithms developed over the years, but so far each has only been applicable for a specific movement catalog. SRS are exceedingly high dimensional planning problems, yet there is a common problem structure; all subunits have identical local motion constraints. Clearly this structure is exploitable, evident by the existence of efficient shape-to-shape planning algorithms. Yet, we lack a formal understanding of the structure that has been exploited by the plethora of results. Perhaps there might exist a general SRS motion planner for lattice based SRS, that could be applied to a large variety of SRSs, that could scale to tens of thousands of subunits?

In looking for a general solver for shape-to-shape reconfigurations tasks, we have found that existing planning methodologies used in other areas of robotics simply won't scale to the numbers of subunits necessary. This no room for error when planning in very high dimensional spaces, backtracking, and therefore search, is impractical. There is a difficulty in predicting the long range motion opportunities from a particular state, because the constrained local motions of subunits can cause bottlenecks in the state space and jam a planning attempt. We have discovered though,

that while some state spaces appear to be inherently difficult to plan with, there is the possibility of restricting the state space in such a way as to expose a subspace whose potential for planning is better.

We found that whether or not a state space may support efficient planning can be determined by finding an ordering over state spaces generated by adding a subunit. While a general state space may not posses the ordering property, a restriction might do. Meta-modularization restricts planning choice branches to reside on state space vertices corresponding to specific arrangement of meta-modules; the surface model restricts planning choice branches corresponding to configurations that do not posses 1-wide tunnels of empty space. Both models, when the plan is executed, use sequences of the original motion catalogs to move between these specially selected configurations or vertices of the state space graph. The minor ordering observation itself is not so much about the process of planning between vertices, but the selection of feasible vertices from the more general state space. We have shown that certain state space vertex restrictions lead to state spaces that are more desirable from a planning perspective, and that a proposed vertex restriction can be evaluated using global graph descriptors.

With a mathematical description of the state space properties we believe are desirable in an SRS planning problem, it may become possible to automate the processes of finding these state spaces within a given SRS problem description. This could be used as an initial step to creating an SRS motion planning compiler; a program that builds a planner that can efficiently solve shape-to-shape planning problems for arbitrary set of SRS motion constraints. We discuss some of the potential issues in section 8.

While this thesis has extensively used the hexagonal metamorphic robot as a model of self-reconfiguration, the goal was to progress towards developing deeper understanding of the reconfiguration problem in general. The HMR is not a particularly useful robot to construct in an engineering sense. Self-reconfiguring robots that have been constructed that show promise in real engineering are systems such as M-TRAN [55]. However, developing motion planners for these systems has so far been difficult because the local motion constraints are unnatural for a human to visualize. Removing human input into the development of motion planning algorithms appears to be a necessary step to be able to utilize self-reconfigurable technology in pragmatic instantiations.

SRS systems, in most forms, can be abstractly represented (or approximated) as relabeling operations performed on a lattice. With this view, the differences between different SRS architectures are less pronounced. Our main contribution was characterization of SRS definitions that have special properties with implied computational benefits, and that these special SRSs can be found within the non-special SRSs. We are quite hopeful that this insight can be utilized to discover efficient planners for complex SRSs, such as M-TRAN, without the use of human intuition.

We might ask whether these insights are useful outside of the SRS domain in a general planning context. High dimensional planning problem are typically associated with "the curse of dimensionality", yet in the SRS case it is possible to find plans in polynomially bound time. While the graph minor ordering of the state space may be too specialized an observation for problems outside the

SRS domain, its implications of dividing a hard, high dimensional problem into a series of incremental, recursively solvable problems is not. It is worth pondering whether the standard textbook representation of a machine learning problem as fixed length vector is the best representation for all classes of problems. We have observed that an important regularity in the SRS domain is found when contrasting the state spaces induced by adding a subunit (a DOF), and it is that property that allows an efficient planner to scale to high dimensions. Similarly, when a problem can be posed as a variable dimensioned problem, perhaps we should look for high level correlations as the problem size increases.

8. Future Work

%CHANGE2B%

An essential ingredient for the utilization of SRS technology is shape changing control strategy. There have been a number of efficient motion planning algorithms published in the literature. However, each of these can only be applied to the SRS it was designed for. Furthermore, the most promising SRSs from an applications point of view tend to be too complicated for researchers to develop motion planning algorithms for (such as M-TRAN). There is a saying in the SRS community, "if you can build it, you can't control it. If you can control it, you don't want to build it", reflecting the conflict between what motion planning research has achieved and what hardware is needed to do traditional robot tasks. Given that future practical instantiations of SRSs are likely have a very complex set of motion constraints, the hand designing of motion planning algorithms is unlikely to scale to the challenges ahead. We need more general methods of development of motion planning algorithms for SRS for the technology to flourish.

The few general attempts at motion planning have under-utilized the available moves in the underlying state space which results in inefficient planners [7, 22]. Until now, we have not understood what state space structures efficient motion planning algorithms have been exploiting to to make the observed gains in a very high dimensional planning problem. Now that we have have a handle on what makes a planning space good for planning (chapter 6), and that we know that these spaces can be found inside hard planning space, there is the potential for mining a hard state space to find an easy planning space, and then automatically generating a planner for the discovered state space. This approach potentially could scale to much more complex SRS definitions than would be achievable by using human insight alone.

While the main contribution of this thesis was defining exactly what an easy planning state space is, we have further insights on how this observation might be exploited to create an SRS motion planning compiler. Such a compiler would generate a motion planner automatically, when given a specific SRS motion catalog. There are, however, a number of difficulties though which will require further research, but by going through the difficulties, we can put the implications of chapter 6 in context and highlight the next research steps necessary.

The goal of general shape-to-shape reconfiguration planning is to invent a motion planning compiler that, given five ingredients as inputs, will output an efficient SRS planning algorithm. The compiler requires:

1. A definition of the embedding lattice, $\mathcal{L}$
2. The permitted labels on the lattice, the alphabet, Σ
3. The local relabeling options (the motion catalog), C.[2]
4. A seed configuration, O.
5. A relabeling catalog, B, used to build valid configurations from the seed

[2]While it may be desirable to add global constraints as part of the motion model, such as keeping certain labels fully connected at all times, this probably generalizes the problem too far to be tractable in the immediate future.

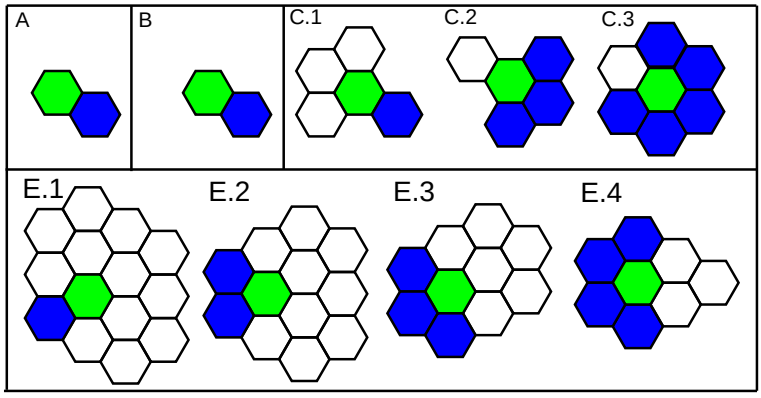


FIGURE 8.1. Build catalogs, A, B, C, D and E represent Chirikjian, Claytronics, Ghrist and surface models build catalogs respectively. Green represents where the new subunit will go, which can only be added if the correct local context is present, white representing empty space and blue another subunit. Chirikjian and Claytronics build catalogs ensure new subunits are added adjacent to an already placed subunit, thus creating connected configurations. Ghrist's build catalog is designed to avoid creating gross changes in topology when building a configuration, and the surface model's prevents one wide tunnels of space in the configuration.

Points 4 and 5 have been overlooked previously, but are an essential ingredient. In the original work of Chirikjian [13], it was specifically stated units must remain connected at all times, which was reinforced through a global constraint. Thus, when building a 5 unit configuration from an anchor, one-at-a-time, it is intuitive that placement of an additional unit must occur adjacent to an already placed node. B describes the sequential options of where relabeling may occur when building a feasible configuration incrementally. O, the anchor defines the first term of any build sequence. For example, Ghrist's [25] motion catalog could not change the gross topology of a configuration, so O must be in the gross topology of problems that are to be tackled, and the build catalog B cannot alter the gross topology as shape-to-shape problem instances are built. Furthermore, for solving surface-to-surface problems, the build catalog must ensure that a constructed configuration does not contain 1-wide intrusions of space. Figure 8.1 shows the build catalogs for the models described in this thesis. The seed in all cases was a single subunit. So it is B and O that can select which vertices are picked from a state space.

The output of an SRS motion planning compiler is a function that is capable of providing a sequence of relabeling operations applied at points of the lattice when given initial and goal labeled lattices, $f : (\mathcal{L} \mapsto \Sigma, \mathcal{L} \mapsto \Sigma) \mapsto (\mathcal{L} \times C)*$. In chapter 6 we discovered that easy planning spaces were well ordered by the graph minor relation and stated that easy state space graph containing n units are a minor of the state space containing $n+1$. Using the terminology from above, this is expressed as, the state space graphs constructed from vertices generated by all possible n applications of B starting from O are graph minors of all $n+1$ applications of B starting from O, where the edges of the graphs are given by (potentially sequences) of applications of C (figure 8.2).

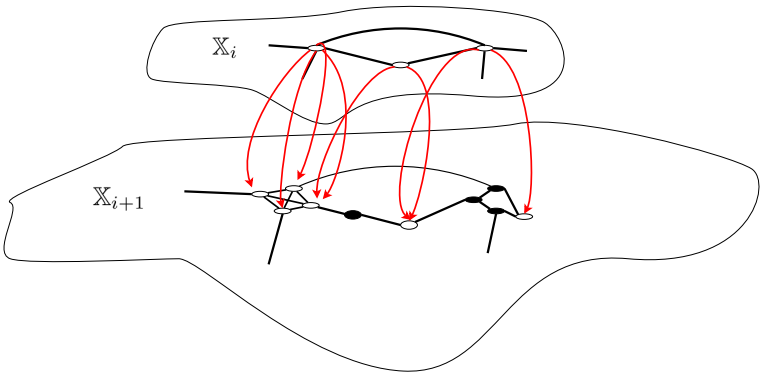


FIGURE 8.2. The build catalog is used the function used to increase the the problem complexity (red); converting one set of planning vertices of one state space, into the planning vertices of the next in the series. The edges between vertices is defined by the motion model, C. There are situations (such as the surface model) where going between planning vertices (white) requires intermediate states (black) which are reached using C but are never states considered as choice points during motion planning.

To utilize the insights of chapter 6, on presentation of an arbitrary SRSs problem definition, it is likely that initially provided state spaces are not well-ordered by the graph minor relation. So the first step is to adjust the catalog B to select vertices for a new SRS motion model that is ordered by the graph minor relationship. The motivation of this new SRS motion model is that long range planning shall be easier than the original definition. The new build catalog that generates the better vertices we shall denote as B'. Ensuring that the new problem is a subspace of the first is simple enough as catalogs are defined as local criteria of what labels must be present for the relabeling to be applied (the support). Therefore, simply adding additional local label requirements to the original supports can only shrink the resultant state space graphs *i.e.* $B' > B$. So a program could incrementally add constraints to the base build catalog B until it finds the fewest necessary constraints to generate a state space family that is ordered by the graph minor relation.

Deciding whether a particular build catalog generate state spaces that are ordered by the graph minor relation is difficult, however, for several reasons. Firstly, proving the existence of a minor ordering would would require generation of an inductive proof and this is likely to be semi-decidable. So instead, one might consider that generating the first few state spaces, and checking for graph minor ordering between them, might be an easier pragmatic route to take. This simpler route, unfortunately, also has difficulties owing to the nature of the SRS problem.

In chapter 6, table 3, we fully expanded the Ghrist and Surface state spaces and found they only differed for $n \geq 5$, by which time the state spaces has over 900 vertices and over 10,000 edges. Checking the graph minor relation holds for two graphs is NP-hard, so checking the graphs at the $n = 5$ would be intractable, let alone building up evidence for larger n. But there is hope.

The graph minor relation is a graph theoretic concept where the vertices are devoid of further meaning. For the SRS case, however, vertices are labeled configurations. In chapter 6 we used

this labeling to provide a notion of locality between vertices of graphs belonging to different state spaces. In our new nomenclature, this relation across state spaces is given by the build catalog, B. Now checking $G \leq H$, when a small set of vertices from H have to be mapped onto G becomes extensively more constrained than in the unlabeled vertex case, to the point where it is potentially tractable to compute. Furthermore, in finding the set of edge contractions, edge deletions and vertex removes to show H is a minor of G will automatically uncover the underlying sequences of SRS moves from C that translate between the vertices selected by B'. Perhaps by providing enough examples of sequences of C that translate between pairs of vertices generated by B', it would be possible to predict the B' neighborhood of an arbitrary vertex generated by B' (using the techniques discussed in appendix 2). This would be important for a motion planning compiler because although the catalog B' generates the vertices of an augmented state space graph, the edges are concatenations of operations from C, and so applying just an individual application of the C catalog to a B' vertex may not generate the easy planning neighborhood, which is essential information for higher level planning.

Supposing that the above is possible, and from an initial SRS definition it is possible to find a new build catalog and that generates a state space with new vertices and edges that are ordered by graph minor. How would one generate a planning algorithm from this, given the new state space would still be massive and intractable to perform searches upon? Let us first perform an abuse of notation to denote what we meant in previous chapters as a state space containing n units. Let $O \times (B')^n$ represent the state space *graph* using vertices generated by applying all combinations of operations from B' n times, and edges being specific concatenations of C as described above. Then $O \times (B')^n \leq O \times (B')^{n+1}$. If we invert the graph minor relation, translating from $O \times (B')^n$ to $O \times (B')^{n+1}$ involves only edge additions, expansions and vertex additions, and critically implied is the face that **edges of $O \times (B')^n$ are never lost**. This means that a plan topology for the n state space will exist in the $n+1$ state space. The sequence of moves in the plan may grow longer at some locations by the action of an edge expansion (the inverse of an edge contraction), but the plan is guaranteed to exist as a direct result of the minor property. Concretely, for the surface model, adding a subunit to the periphery causes other subunits that moved through the previously unoccupied space, now have to deviate around, and the possibility of deviating guaranteed because of the restrictions of where a subunit can be added (an example follows later).

The reuse of plans from the previous state space implies that a motion plan between two $O \times (B')^n$ configurations can be incrementally built up starting from simpler $O \times (B')^1$, $O \times (B')^2$ ... cases. Finding a possible sequence of B' operations to generate a given configuration is straightforward. As the build catalog B would commonly only add labels (ignoring the loss of the infinite unoccupied label), while there may exist multiple solutions for incrementally growing a given configuration, there is no need of backtracking, or of intermediate states, so a solution could be found in $O(n)$ using a greedy procedure. Figure 8.3show two examples of building a 5 unit surface configuration by applying the build catalog of figure 8.2 5 times. Generating the example solutions were easily computed, starting from O, apply any B' move that reduces the distance between the labellings.

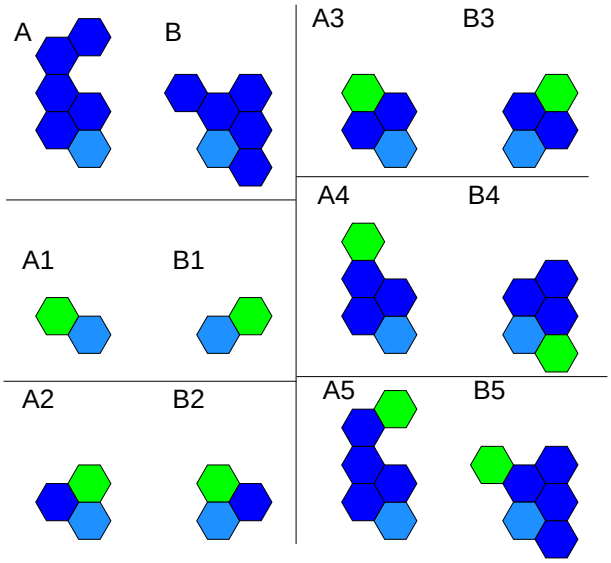


FIGURE 8.3. The surface configurations A and B can be build incrementally from the surface build catalog. A 5 unit configurations requires 5 steps. A motion planning task of changing A to B can be determined by solving each of the sub-motion planning problems, $A1$ to $B1$, $A2$ to $B2$... The lighter blue hexagon represents the seed configurations, the green where a unit was added each increment.

Distance in this context, would be the sum of lattice coordinates whose labels disagree, which generalizes to complex SRS with larger alphabets.

Using the example incremental states in figure 8.3 as examples, we will now show how a motion plan for a $O \times (B')^{k-1}$ can be augmented to create a new plan for the next in the series, $O \times (B')^{k}$. Consider the motion plan between A1 and B1. Only one subunit is movable, so exploring the state space using the motion catalog C will quickly find a solution. Using a search methodology such as iterative deepening depth-first search [67] would find the optimal solution to this case, moving the free subunit of A1 clockwise. This motion planning solution is represented in figure 8.4 as P1.

The next planning problem in the series is forming a plan between the states A2 and B2 by augmenting the plan P1. In our example, the additional subunit is added in the path of the previous plan. The first step to reusing a plan would be to adjust the states in order to accommodate the plan from the previous iteration. As a direct consequence of the graph minor ordering property of nice state spaces, the previous plan topology is guaranteed to exist. For the surface case of SRS, the manifestation of this property is that the P1 motion plan can be redirected around the added subunit. After the plan has been augmented, the second step would be to find a path between the newly labeled areas, but in our example this is trivial because the added subunit was in the same location for both states A2 and B2. Figure 8.4 depicts this new plan as P3.

Solving the motion planning problem with three modules reusing plan P2 is similar to how P2 was elucidated. The first route planned again has to be rerouted around the newly added subunit in the start configuration. After the P2 is executed, it should be obvious that the movement of the

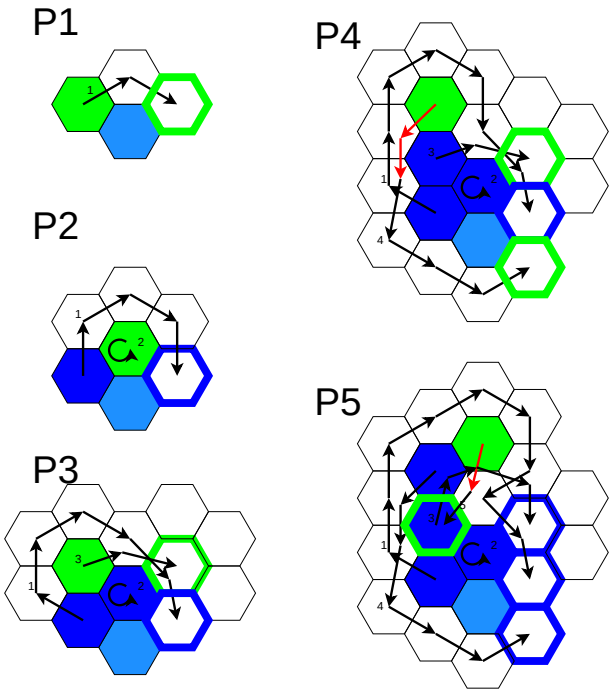


FIGURE 8.4. Solving the 5 unit planning problem of figure 8.3 incrementally. Each iteration the previous plan is augmented by routing paths around the newly added subunit. Motion paths are represented by numbered black arrows. If motion is constrained by the new subunit, as is the case for P4 and P5, then the newly added subunit can execute "get out of the way moves" as and when needed (red arrows). Once the previous plan has been fully adjusted and executed, the newly added subunit (solid green) can be moved into its goal destination (green outline).

third subunit into its goal position is trivial, which completes the plan P3. A new class of difficulty arises when planning for 4 units.

To solve the case when there are 4 units by reusing the P3 plan the initial path of P1 must again be routed around the newly added subunit. P2 represented no movement so can execute too, but the path created in P3 cannot be executed because the newly added subunit was placed on top of the moving P3 subunit. However, recall that in the proof of chapter 6 that in order to recover the previous state space, the new subunit may have to "get out of the way" for the local contexts to not to interfere. In the surface model context, this was guaranteed to exist via moves around the periphery by the new subunit. So for the plan P3 to be executed, the new subunit must move (shown in red). The subunit could either move two steps in either rotation before the new subunit of P3 can start to move. This extra motion planning would not significantly increase the complexity of the motion planning problem, because these "get out of moves" are only considered as and when needed. After the newly added subunit has unblocked the execution of P3, the added subunit can move into its final goal location from its perturbed start location, thus creating P4 of figure 8.4.

Building a plan for P5 from the P4 plan is not significantly different. The "get out of the way" moves of P4 are blocked by the added 5th subunit, but are re-enabled by the 5th subunit executing its own "get out of the way move". The "get out of the way" moves created at the P4 stage have no special meaning when considered during the P5 state. A concern might be that the these "get out the way moves" cascade and grow as the complexity grows, but this is not the case, for the HMR case at least, because the cause is a local interaction whose relative importance diminishes (statistically) as the size of the configurations grow.

Several parallels between this incremental approach to solving reconfiguration planning task and the algorithm of chapter can be drawn. The algorithm of chapter 5 grew a connected $PLACED$ set in line with the solver, which corresponds to an ordered breakdown of configurations through a sequence of build operators. The algorithm of chapter 5 moved subunits from the periphery of the current configuration (i.e. starting outside and working towards the seed location) to locations on the periphery that overlapped with the goal configuration (i.e. nearer the seed location and working away from the seed location). Pairing of subunits is preserved in the algorithm described above, but the ordering is inverted; movement between subunits in the start and goal configurations are started at the seed location and moved outwards. A feature of the above algorithm with no corresponding parallel in the chapter 6 is the "get out of the way" moves, which explains why the Surface-to-Surface planner got stuck more frequently at smaller problem instances.

The rerouting of previous plans, and the "get out of the way" moves are guaranteed to exist through properties implied by the graph minor ordering of successive state spaces. So the opportunity to solve planning tasks recursively is afforded by the graph minor property observed in nice planning spaces. We found that while many SRS definitions don't possess this property, a subspace does. Subspaces of a larger state space can be selected by adding addition constraints to a base build catalog.

The goal of general SRS motion planning though, is forming motion plans in the entire state space, not just a subspace. In chapter 5 we developed a second planner, the Claytronics-to-surface planner that complemented the surface-to-surface planner in order to solve a larger proportion of the target Claytronics state space. While the graph minor observation of chapter 6 suggests the existence of a "highway" subspace suitable for long range planning, what of the remaining volume of the target problem?

Fruitful research may be found by exploring the meaning of the build catalog further. The vertices of the efficient planning space are constructed from n applications of the efficient build catalog B'. The vertices of harder portions of the planning space therefore must be build from $iB' + kB$ build operations, where $i + j = n$. This can be visualized diagrammatically as shown in figure 8.5. The number of applications of the B catalog provides a convenient distance that a configuration is from the desirable portion of the state space for planning, and could also represent a measure of difficulty of a configuration is to plan with.

In chapter 5 the Claytronics-to-surface planner operated by excavating subunits to ensure a two wide tunnel connected all regions of unoccupied space. Only those subunits that lay on the two tunnel path were classified as mobile, and once they were moved were no longer considered. Using

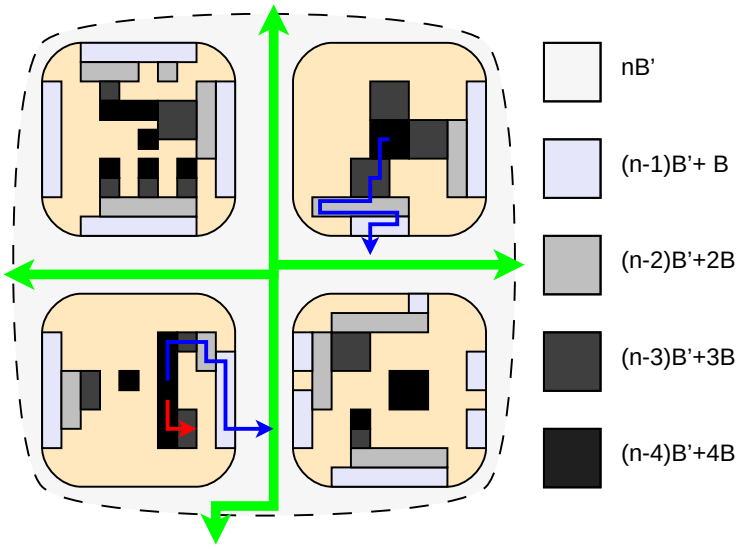


FIGURE 8.5. Visualization of the state spaces of SRSs. Configurations built using only the B' catalog have the nice minor ordering property. This implies long range planning across a large volume of the state space is possible, indicated by the green routes. Adjacent to those configurations are those that require one application of the basic B catalog to construct (and adjacent to those two B applications *etc.*). The state space outside of the nB' areas is messy, but the number of applications of B suggest a gradient that can be search to get into nB' territory, shown in blue. Due to the unpredictability of the state space outside of the nB' volume, that gradient may lead to a local minima, shown in red. The frequency of red routes is likely to be SRS dependent. If a motion planner did not restrict itself to the nB' volume when long range planning, it is likely to get stuck in the nearby messy parts of the state space, where bottlenecks are frequent.

the build catalog nomenclature, those subunits correspond to necessary applications of the B build catalog after all possible applications of B' have been exhausted. Moving a subunit out of the two tunnel path to a safe area elsewhere creates a configuration that requires one less B operation to construct.

The existence of a gradient representing the distance from the desired efficient part of the state space clearly has computational advantages when forming motion plans in the more general state space. The distance provides a clear direction for a planner optimize towards, and provides sub-goals to consider one search at a time, thus hugely reducing the computational complexity of the overall search for a path towards a graph minor portion of the state space. However, in the algorithm of chapter 5, the planner did get stuck more often as the problem size increased, albeit slowly for the Claytronics case. A concern is that for different models of SRSs the behavior outside of the nice planning space might be much worse, and using the B operations as a heuristic maybe not be feasible. Certainly, even in the Claytronics case, there are no guarantees of successfully solving

all tasks outside of the minor ordered portion of the state space. That said, it is our opinion that an algorithm that tries to solve a large portion of the state space and either succeeds or fails in polynomial time is an improvement over an algorithm that always successfully solves a smaller portion of the planning space in equivalent time (both approaches being more desirable than a semi-decidable planning algorithm).

Appendix 1: Random configuration generation

In this appendix we explain the design decisions taken for the random configuration generators. Generating random configurations is far from trivial, but regarded as largely off-topic for this thesis to address properly. The solution presented here was a pragmatic partial step, to which there is (as far as we are aware) no completely satisfying solution in general.

Our main use of randomly generated configurations was to determine the average time complexity of motion planning algorithms. In general, to determine the average behavior of any algorithm, one must try **all** possible combinations of inputs and normalize. For many algorithms, the number of combinations make determining the true average unfeasible, so instead, to reduce the number of experiments, the average behavior is approximated through sampling.

The statistically sound approach to estimate an average is to generated **uniformly** distributed inputs. In the domain of self-reconfiguring systems, this means generating **uniformly** distributed configurations. An obvious interpretation for uniformity, in this context, is that every possible configuration should have an equal probability of being generated (over the set of all possible valid configurations). There is however, a technical difficulty in randomly generating configurations adhering to a uniform distribution (more on this later). Previous works have avoided the difficulty in different ways. Some studies have used hand crafted configurations to test motion planning algorithm [58]. This approach has the drawback of not scaling to large numbers of trials, introducing human bias and introducing publication bias. Other works focus instead on worse case analysis [65] of a motion planning algorithms, but many planning algorithms have identical worst case behavior yet different average case behavior (examples of this behavior outside of self-reconfiguring systems include well known algorithms such as quick-sort). Algorithms that are optimized purely against worse-case performance lose sight of the basic goal of SRS planning, *get to the goal configuration in as few moves as possible*, which average case behavior is a better proxy for. Thus, throughout this thesis, we elected to randomly generate configurations in order to estimate average case behavior for motion planning algorithms, but with a caveat that we could not generate statistically sound uniformly distributed configurations.

The problem is that in order to **uniformly** generate a configuration, at the end of the generation process, the final configuration chosen must have had an equal probability to have been selected as any other possible valid configuration. This criteria is very difficult to obtain in a computationally feasible manner, given the operations available to create valid configurations (be they a Claytronics, Ghrist or surface configuration). The only work we know of that has nearly approached this subject is [50]. Martins *et al.* used group theory to identify graph orbits representing automorphisms. The graph orbits determined different locations on the periphery of the growing configuration which lead to unique outcomes given shape isomorphisms. Then they could enumerated configurations whilst avoiding placement of a unit in the same orbit that would lead to the same shape class under isomorphisms (figure 1.6). There work suggested this saved computation time, but interesting automorphisms become asymptotically rare as the number of units grow (e.g. the are almost no symmetric very large configurations relative to the space of all very large configurations). The

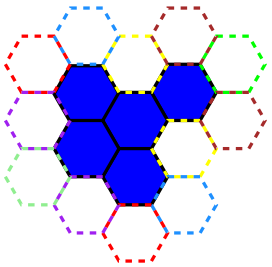


FIGURE 1.6. Martins *et al.* segmented peripheral locations into groups, reflecting equivalence of a location under automorphisms. Periphery nodes are dashed in a color that represents membership to a group.

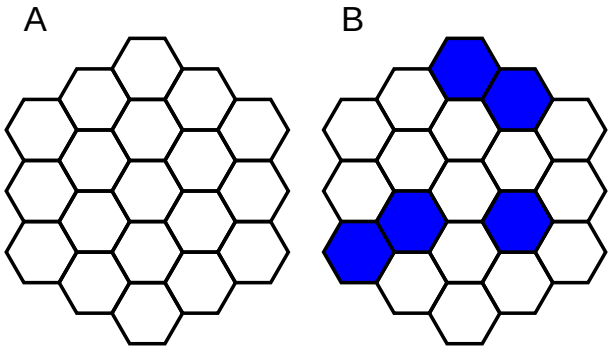


FIGURE 1.7. If n distinct locations are randomly selected in a hexagon shape of diameter n, the probability of generating a connected set of locations rapidly diminishes. However, all possible connected sets of locations have an equal probability of being generated, but not in polynomial time.

process does save enumerating the 6 rotational symmetries, but there are simple, faster and computational cheaper ways of excluding these particular class of isomorphisms. Thus, gains using their method can only be made when generating small configurations. So sampling through enumeration has a time complexity of $O(k^n)$.

Many other possible methods of generating uniformly distributed random configurations are also not polynomially bound. A simple example is using the prior knowledge that an n unit configurations has a diameter less than or equal to n. To achieve uniform chance, one could repeatedly distribute n modules in a hexagonal space of diameter n according to a uniform distribution, and select the first configuration that was valid. Unfortunately as n grows, the probability that a valid configuration is generated rapidly diminishes. It is becomes clear, regardless of the nuances of each catalog, that the probability of generating a set of units that are also connected approaches 0 fast.

The pragmatic approach that was undertaken to ensure configurations were generated in polynomial time was to build up a configuration incrementally. Starting from a seed location, $n-1$ units are added to the periphery in such a way as to ensure a valid configuration is the end result at each step. The definition of a valid configuration, of course, varies with the catalog being used. Placing units on the periphery ensure the growing configuration at each step is connected, a global

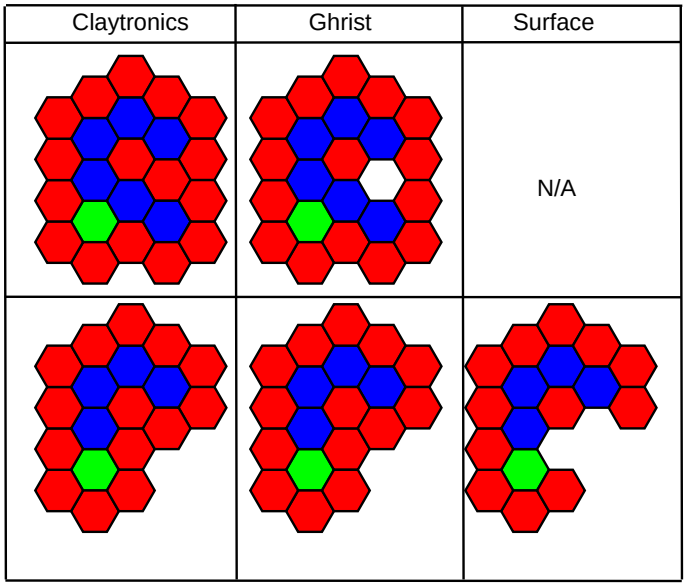


FIGURE 1.8. In the diagrams above, we show the potential periphery locations (red), where a unit can be added to create a valid configuration for the models considered in this thesis. Note that in the top row, the arrangement of seed (green) and already placed units (blue) are not a valid Surface configuration to start with.

constraint common to all models. Determining whether or not placement at a particular periphery location results in a valid configuration only requires checking the local state of locations around the candidate location (as the only global constraint of connectivity has already been satisfied). The exact local requirements for each model is precisely described by the build catalogs discussed in chapter 8. The overall constructive approach to building a n unit configuration from a seed is as follows:-

initialization let the current configuration be just the seed module
1 while the current configurations size is less than n
2 select a peripheral location, p that is adjacent to a unit in the current configuration
3 If placement of a unit at p in the current configuration produces a valid configuration, modify the current configuration with an addition of a unit at p. Loop

1.1. **Generator 1.** The method for selecting the periphery location, p differed from experiments performed during (chapter 4) to the later experiments in (chapter 5) . In earlier experiments the random generation algorithm iterated as follows:-

1. a unit from the current configuration was uniformly selected, b.
2. One of the six adjacent neighbors of b was uniformly selected, p (i.e. $dist(p, q)$=1).
3. If the addition of a unit to the current configuration at location p results in a a valid configuration, then the current configuration is updated. Otherwise the current configuration remains as it is.

Placement attempts repeated until an n unit configuration was constructed. This approach was later found not to scale to large configurations, as generally a random selection of b tended not to have a valid neighbor for incremental placement (the test in 3 failed more often as n increased). This is a manifestation of the property that the number of peripheral nodes scale at $\sqrt{n}$. In large configurations, most units are fully surrounded by other units. This was not an issue in the early experiments where the motion planning algorithms were inefficient and experiments were run on configurations containing less than 30 units. In small configurations (such as in figure 1.8) periphery nodes actually outnumber placed nodes.

1.2. **Generator 2.** In the experiments of chapter 5 and onwards, configurations containing thousands of units were required to evaluate performance. The random configuration generator modified so that it did not suffer the same performance problems. A new set, B was maintained, that contained location's of units that were blacklisted for random selection. Locations were blacklisted when the random selection system discovered that the unit was surrounded. Due to the nature of incremental building i.e. never removing units, once a unit has no valid adjacent locations for building, there exists no combinations of adding units that will change a blacklisted location to an un-blacklisted location. So the new algorithm iterated.

1. a unit from the current configuration, not in B, was uniformly selected, b.
2. **All** of the six neighbors adjacent to b were tested for validity of placement.
3. if none of the neighbors could have a unit placed, a new location was added to B, $B_{new} = B+p$.

Otherwise one of the valid neighborly locations were selected to add to the current configuration.

The effect of the blacklisted set B is to prune the choices in b that previously led to a monotonically increasing risk of b not having any neighbors with valid locations adjacent. This modified random configuration generator ran at $O(n)$, although the implementation details are perhaps somewhat clumsy.

1.3. **Candidate Generator 3.** Perhaps an simpler $O(n)$ random generator for sake of theoretical arguments algorithm would be the following. Observe that the set of valid periphery locations for building changes only in the spatially local area where a new unit was added. The set of valid periphery nodes for building is the same as the $GROW$ set of chapter 5 for Surface configurations. Updating the set of potential valid build locations is an $O(1)$ operation, using data structure described in chapter 5. The new random configuration then simply uniformly selected a valid build location (red locations in figure 1.8) and updated the periphery build set n times to build an n unit configuration in $O(n)$ time and space.

However, the above algorithm wasn't implemented because it does not work on Ghrist configurations naively without development of a different mechanism for recognizing local changes. Development of a the third random configuration generator was not regarded as worthwhile because **all** of the generators described above generate non-uniform distributions of configurations, so its difficult to fundamentally recommend one generator implementation over another. Tinkering with random configuration generator distributions does not alter the gross performance of motion planning algorithms. If an algorithm suffers from exponential bottlenecks in the search space, no

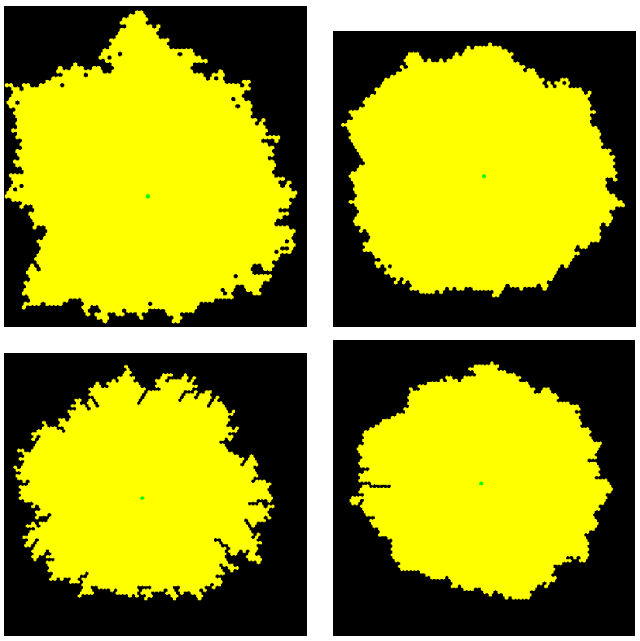


FIGURE 1.9. Top: Two randomly generated Claytronic configurations containing 4096 units. The leftmost configurations comes from the generator used in chapter 4. The second configuration comes from the generator of chapter 5 onwards. Enclosed space can be seen in both. Bottom: Two randomly generated Ghrist configurations, using the same two generators. For the Ghrist catalog, unit movement around the perimeter is blocked by 1 wide pockets in the surface. While the leftmost configuration contains more of these surface defects, the rightmost also contains several examples of this major source of bottlenecks. Ghrist configurations cannot contain enclosed space by construction.

amount of tweaking with the random configuration generator will significantly help. Difficult cases are generated by all the generators described above (figure 1.9), and so we conclude the generators used were sufficient for estimating average case behavior (particularly for estimating the gross asymptotic behavior).

1.4. **Discussion.** The source of bias in the random generators is that multiple execution paths can lead to generating a specific configuration. Consider the two unit configurations in figure 1.10 A and B. If the third random generator configuration generator described above was used, it has two ways of generating the configuration in figure 1.10, B, verses only one way of generating the configuration in figure 1.10, A. The bias of a specific configuration is the number of unique depth first searches (DFS) that can be performed on the connectivity graph starting at the seed (figure 1.10, bottom), $|DFS|$. As the bias factor can be explicitly calculated, the generator can be debiased by introducing an acceptance probability for a configuration, $a = 1/|DFS|$. This observation does not help development of a practical uniform random configuration generator though, because $|DFS|$ grows exponentially with n. Perhaps limiting the choices of the growing DFS, or interleaving

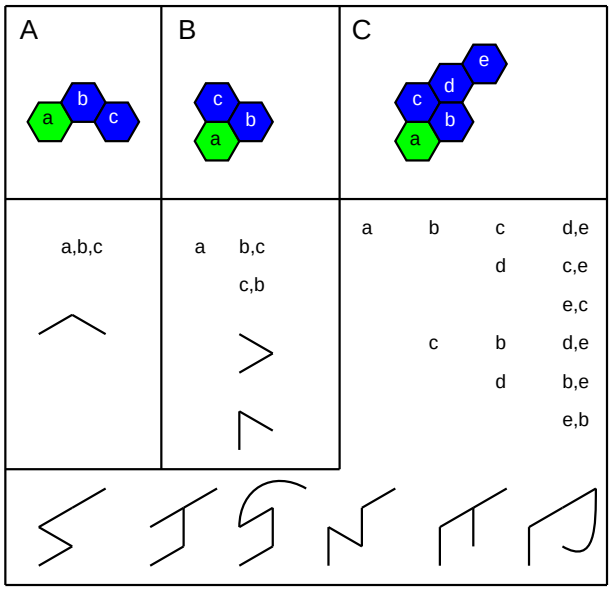


FIGURE 1.10. Configurations can be generated incrementally from a seed in different ways, depending on the internal connectivity of the shape. Configuration A can only be generated by one path starting from the seed. There are size different ways configuration C can be generated.

acceptance probabilities into the inner loop of incremental random configuration generation could lead to a fruitful discovery, but we consider this out of scope for the work presented in this thesis. It is mentioned as a useful starting point for future research in to generation of random configurations.

Additional utility for further research may be found in a closely related previously studied problem. Polyominoes are a generalization of dominoes. An n-polyomino is a connected set of n square tiles on a square lattice. In popular culture, seven tetrominos (4-polyominoes) were used in the well known computer game puzzle Tetris. Tiling the infinite plane with a finite set of polyominoes is equivalent to Wang's domino problem [28]. Enumerating the number of n-polyominoes for given n has been studied, but no polynomial bound algorithm has been discovered yet . However, asymptotic growth has been approximated and bounded in both directions. The number of polyominoes of size n conceptually relates strongly with the number of possible configurations there are for n number of units[3] *i.e.* the cardinality of the size of the set we wish to uniformly sample from. A bounded and approximated value could be useful for debasing a random configuration generator or useful in other proofs.

In summary, generating uniform samples from the set of all possible valid configurations is a very difficult problem, perhaps unsolvable in polynomial time. We developed a pragmatic bias solution for generating random configurations instead. We believe that when the bias generators are used to

[3]although the mathematics needs to be transformed for use on a hexagonal lattice, and local constraints may need to be considered

construct random motion planning tasks, they are capable of characterizing average case behavior of motion planning algorithms, but this cannot be proved without further research.

Appendix 2: The C4.5 data mining algorithm and its application

A persistent issue when developing novel data structures, novel motion planning algorithms and new proofs in the SRS domain is bugs. In order to create efficient algorithms, numerous cases of low level scenarios must be handled correctly in order to make computational gains. There is often a nagging doubt that while the current implementation *seems* to work, there may an unexpected situation that invalidates the logic. This is compounded by the fact that unusual cases are rare in the state space, so enumerating cases by hand is unfeasible.

This problem was particularly acute in development of the proof in chapter 6. The surface state space in earlier chapters was defined by describing what local constraints were not permitted in the tour around the empty perimeter locations. In the implementation, the constraints were enforced by using a custom data structure. However, for the proof of chapter 6, the high level description of the state space had to be translated into a set of local constraints that was equivalent.

The initial draft of the proof was developed by hand, which was accepted as valid by the peer review process. It was not until addressing an unrelated problem that it was discovered the translated representation of the Surface constraints were actually incorrect. It was at this point a new method was developed for aiding the process of extracting salient cases from a large data set of examples. The ID3/C4.5 algorithm was found to be an excellent fit for this particular (classification) problem. In this appendix we shall explain in detail how it was applied.

2.5. **The ID3 algorithm.** The C4.5 algorithm[62] is an evolutionary improvement to the iterative dichotomizer 3 algorithm (ID3) [61]. The improvements of the C4.5 algorithm over the ID3 algorithm are distracting for the purposes for which we used the algorithm. The C4.5 algorithm was primarily used because it is readily available in the mature Weka 3 machine learning library [29]. The Weka 3 library has been well tested, and provides a convenient GUI for interpretation of the results. The ID3 algorithm and related algorithms are binary decision tree classifiers.

ID3 classifies a datum of attributes $\mathrm{x} = \{a | a \in \mathbb{A}\}$, as being a positive or negative example, $ID3 : \{a | a \in \mathbb{A}\} \mapsto \{+, -\}$. The classification process queries a datum's attributes in the order of a binary tree until a leaf node is reached. Each leaf node of the decision represents a conclusion of what class a proposed input vector belongs too. The decision tree is created by running the training algorithm on a data set of labeled examples *i.e.* it's a supervised learning algorithm.

The ID3 training algorithm is an recursive algorithm that starts with all training data as currently unclassified. Each iteration the unclassified set is split into 2 smaller sets which are heuristically nearer to being classified correctly. Each iteration the information entropy of a set is measured according to:

$$E(S) = -\sum_{j=1}^{\mathbb{A}} f_S(j) log_2 f_S(j)$$

The information entropy of a set, S, is measured by counting the frequency of a particular attribute, $j \in \mathbb{A}$, in a set using the frequency function $f_S(j)$. The training algorithm determines

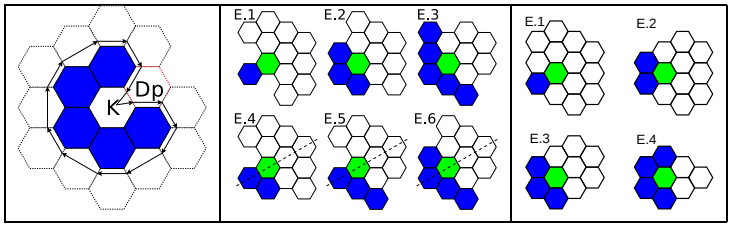


FIGURE 2.11. Left, the dual path and kink defects that are banned from Surface adhering configurations are easily recognizable when a path around the permitted is drawn. Middle, an early incorrect attempt in translating the surface constraints into a purely local incremental build notation. Right, the correct incremental build catalog that implicitly enforces the surface constraints, elucidated with the aid of the C4.5 algorithm (note it's more compact as well as being correct).

an attribute to split, $b \in \mathbb{A}$, the currently unclassified set into two smaller sets by maximizing the information gain, $G(S, b)$ according to

$$G(S, b) = E(S) - f_S(b)E(\{z|b \in z \in S\})$$

If the labeled data of the training set says all of an unclassified set is either a positive or a negative example, then the training algorithm can halt for that branch of the training process and insert a leaf node into the decision tree.

2.6. **Application.** Our use of the ID3/C4.5 algorithm was as an aid in translating from one representation of constraints into another purely local one. Recall that in chapter 5 a surface configuration was defined as a refinement of the Ghrist catalog, with the added constraints that a Hamiltonian path around the perimeter could not contain a kink violation, or a dual path violation (figure 2.11, left). This representation lent itself to a persistent implementation in code, but was hard to utilize in a subsequent proof. Translating these constraints into a local incremental build catalog was found to be error prone when done by hand (figure 2.11, middle). When we used the C4.5 algorithm to help automate the process, the resultant representation was correct, and was more compact than we managed previously by hand. It is for this reason we think the C4.5 algorithm has additional uses in other processes we found error prone. We shall now step through the application of the C4.5 algorithm to this particular problem.

The main ingredient used was the data structure for determining whether or not a configuration was a member of the surface configuration. This is described in detail in chapter 5 but for the purposes described here it simply maps a Ghrist adhering configuration to a Boolean, $isSurface : \mathbb{G} \mapsto \mathbb{B}$. At the time of development, we already had a robust method for building a Ghrist configuration incrementally [25].

The overall method we used for translation was to generate lots of Surface adhering configurations around an empty test location, and then use to $isSurface$ to find out whether or not the configuration remained a Surface configuration when a subunit was added to the empty test location. If $isSurface$ returned true, then the local area around the empty location was presented

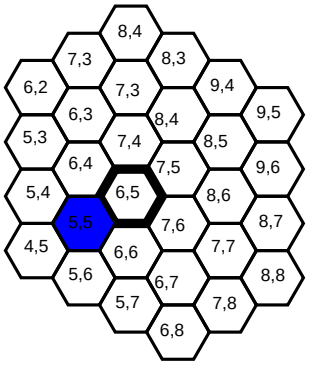


FIGURE 2.12. The context of hex coordinate (6,5) was studied to find out what affected valid placement. One neighbor must always be occupied in an incremental build catalog, so there was no loss of generality by always having (5,5) occupied.

to the C4.5 algorithm as a positive example, and if not then as a negative example. The C4.5 algorithm would then be able to learn a compact representation of what the local requirements are for a subunit to be added whilst respecting the Surface constraints.

The C4.5 algorithm in the Weka machine learning library requires the data to be presented as a fixed length vector classification problem. We chose a subset of positions on the hex lattice and represented these locations as Boolean variable in an arbitrary (but fixed for the experiment) order. The choice of lattice locations were a subset of positions that were found at a distance of three or less from a test location found at hex coord (6,5). It was the locale of the (6,5) coordinate that would be studied to see what surround affected valid Surface placement. As all valid placements in an incremental build catalog must always be next to an already placed subunit, an arbitrary neighbor of (6,5) was chosen to be always occupied, which was (5,5). Locations of a distance of three behind the occupied location (5,5) were known to be unimportant to the experiment. This was the initial setup before further subunits were added and is shown in figure 2.12.

All combinations of adding eight subunits into the study area (30 locations less the central location) were enumerated. It was found that seven or less did not generate enough examples. Added subunits were always placed adjacent to an already placed subunit. If a configuration was a valid surface configuration according to $isSurface$, then that configuration was serialized into a length 30 binary vector. A subunit was then added to (6,5) and $isSurface$ was called again. If the second call returned true, the vector was labeled a positive example of a context that permits an additional subunit to be added whilst preserving the Surface constraints (VALID). If the second call to $isSurface$ returned false the vector was labeled as a negative example (INVALID). 9601 different local contexts were generated, and the list of classified vectors was processed by the C4.5 algorithm to generate a decision tree[4].

2.7. **Results.** The C4.5 Weka implementation was able to fit a compact decision tree that could successfully determine whether or not a subunit could be added at position (6,5) whilst preserving

[4]Weka has cross validation turned on by default, this needs to be switched off

the Surface constraints. Furthermore, while the input vectors were 30 dimensional, the classifier learnt that only a subset of the 30 positions were relevant (figure 2.13).

The first striking about the generated decision tree is that an occupancy of a subunit at (7,5) immediately implies that a subunit at (6,5) can be added, as shown by the root node. This is true regardless of what the status is of other locations around (6,5). Further analysis reveals that this is a special case of a broader pattern, so the tree requires subsequent analysis by hand in order to draw out the broadest principles. To aid readability, the tree was redrawn graphically (figure 2.13). Note that at nodes deeper in the tree, the information from the ancestors represents truisms known about the current state of a queried configuration. So for example, at the second decision node the occupancy of position (6,4) is used to split cases, but at that position occupancy of (7,5) has already been asserted as occupied. When the tree was represented graphically, each node's ancestor information is included .

One issue with the decision tree representation was that we were not looking for a series of conditionals to represent the constraints, rather, we were looking for a small set of cases that captured equivalent information. An addition problem was that the procedure was unable to distinguish isomorphisms. It was noted that the leaf cases whereby the location at (6,5) was found to violate the surface constraints (red) outnumbered the cases where it didn't (green). So we concentrated on the red cases first.

Drawing just the leaves of the red cases revealed that all red cases were rotations of two base cases embellished with further constraints (figure 2.14). After studying green cases in the complete decision tree (figure 2.13), it was found that no green case contained the simplified red bases cases. Therefore, we could conclude that the two red base cases succinctly represent what local constraints are not permitted when incrementally building a Surface configuration. Those two cases very neatly represent that 1 wide intrusions of unoccupied space are forbidden by the Surface catalog (the dual path violations). In fact, they also show that the kink violation is actually a redundant constraint whose explicit representation was unnecessary.

Green cases in the graphical depiction of the decision tree described what state the neighborhood was in scenarios that permitted a a subunit to be added without compromising the Surface constraints. The leaves in the green cases often had redundant constraints inherited from information further up the tree, *e.g.* the right child of both (6,6) decision nodes on figure 2.13. These were easily identifiable because they are asymmetric, and we know from the problem that there is no asymmetry in the constraints. With the ten green leaf nodes laid out graphically, it is relatively easy to pair leaves up to distinguish a smaller set of base cases, as in figure 2.15.

2.8. **Conclusion.** The green base cases describe an incremental procedure of where a subunit can be added to a growing configuration whilst maintaining the Surface constraints. This representation was data mined from a set of examples generated by a very different expression of the Surface constraints. Performing the conversion between representations had already been found to be error prone, but by automating much of the process, a set of nearly 10,000 examples was summarized into a decision tree containing just 25 leaf nodes. The C4.5 algorithm automatically identified

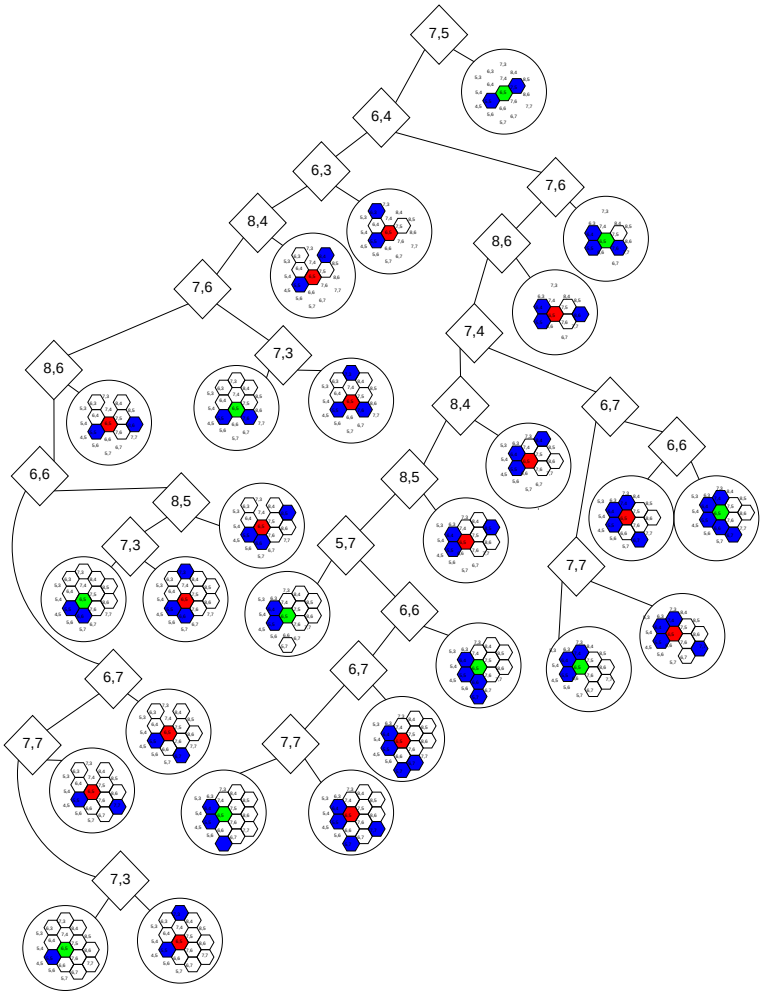


FIGURE 2.13. The complete learnt decision tree for deciding whether (6,5) was buildable (green) or not (red) according to the Surface constraints. In the diagram, the diamonds denote a conditional on the occupancy at the labeled location. The convention used is if the outcome of the occupancy is true, the children are drawn to the right.

that only a subset of the information provided was relevant, and the leaf set was small enough to be summarized by hand without error. As a result, we have high confidence that the base cases elucidated are both a correct and a compact representation of the Surface constraints.

Whilst for the proof we did not want a tree representation of the constraints, a tree is an efficient representation for checking constraints on a computer. It is for this reason we think the C4.5 algorithm could be a useful tool for development of SRS simulators. Inefficient, high level descriptions of constraints can be translated into efficient low level tree representations automatically, and without bugs. This could greatly reduce development time and simply communication of new SRS ideas.

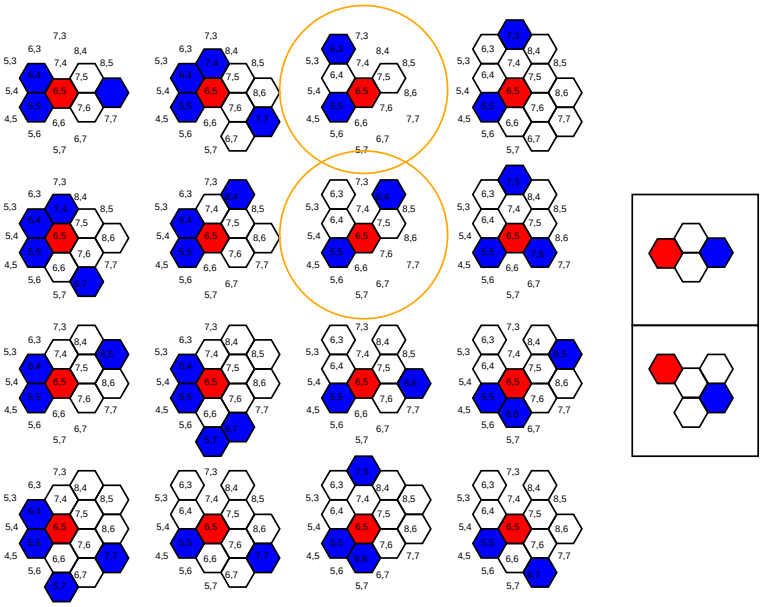


FIGURE 2.14. Nearly all the leaves of the decision tree in figure 2.13 contain the two base cases shown boxed on the right. The two exceptions are circled in orange. Due to the data generation protocol of extending position (5,5) with 8 connected subunits in a limited area, (5,4) and (5,3) must be occupied for the two exceptions. Furthermore, because the configuration is a valid surface adhering configuration before placement at (6,5), it is implicitly implied that position (7,4) is empty too. With (7,4) determined as unoccupied, the two circled exceptions also include the upper of the two base cases.

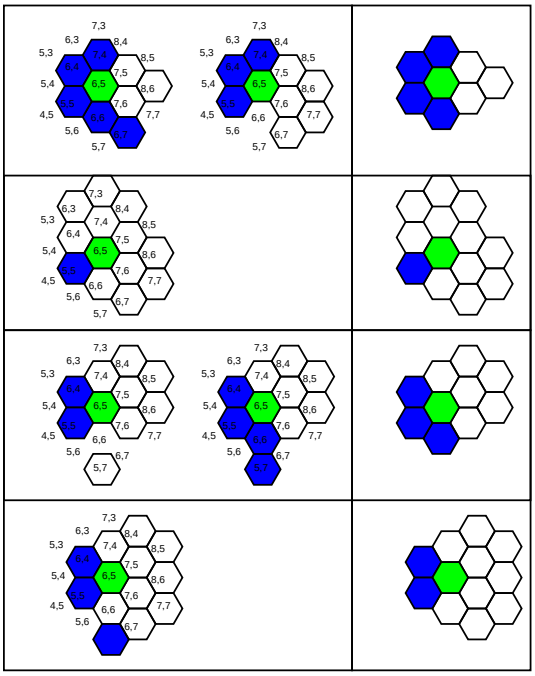


FIGURE 2.15. Base cases of valid build location contexts for the Surface model should be symmetrical. Unioning together leaves of the decision tree (eight of ten shown in the leftmost columns) and trimming asymmetric constraints reveals four bases cases (right column). The top case was overlooked in the first draft of the proof of chapter 6.

References

[1] A. Abrams and R. Ghrist. State complexes for metamorphic robot systems. *Intl. J. of Robotics Research*, 23(7):809–824, 2004.

[2] G. Aloupis, S. Collette, M. Damian, E. D. Demaine, R. Flatland, S. Langerman, J. O'Rourke, S. Ramaswami, V. Sacristán, and S. Wuhrer. Linear reconfiguration of cube-style modular robots. *Comput. Geom. Theory Appl.*, 42(6-7):652–663, 2009.

[3] D. Brandt. Comparison of A* and RRT-Connect motion planning techniques for self-reconfiguration planning. *Intelligent Robots and Systems, Proceeding, IEEE Int. Conf. on (IROS)*, pages 892–897, 2006.

[4] D. Brandt and D. J. Christensen. A new meta-module for controlling large sheets of atron modules. In *Intelligent Robots and Systems, Proceeding, IEEE Int. Conf. on (IROS)*, pages 2375–2380, 2007.

[5] Z. Butler, S. Byrnes, and D. Rus. Distributed motion planning for modular robots with unit-compressible modules. *Intelligent Robots and Systems, Proceeding, IEEE Int. Conf. on (IROS)*, 2:790–796, 2001.

[6] Z. Butler, K. Kotay, D. Rus, and K. Tomita. Cellular automata for decentralized control of self-reconfigurable robots. In *Robotics and Automation, Workshop on Modular Robots, Proceedings, IEEE Int. Conf. on (ICRA)*, pages 21–26, 2001.

[7] Z. Butler, K. Kotay, D. Rus, and K. Tomita. Generic decentralized control for a class of self-reconfigurable robots. In *Robotics and Automation, Proceedings, IEEE Int. Conf. on (ICRA)*, pages 809–816, 2002.

[8] G. Carpaneto, S. Martello, and P. Toth. Algorithms and codes for the assignment problem. *Annals of Operations Research*, 13:191–223, Dec 1988.

[9] A. Casal and M. Yim. Self-reconfiguration planning for a class of modular robots. In *Intelligent Robots and Systems, Proceeding, IEEE Int. Conf. on (IROS)*, pages 246–257, 1999.

[10] A. Castano, W.-M. Shen, and P. Will. CONRO: Towards deployable robots with inter-robots metamorphic capabilities. *Autonomous Robots*, 8(3):309–324, June 2000.

[11] C.-J. Chiang and G. S. Chirikjian. Modular robot motion planning using similarity metrics. *Autonomous Robots*, 10(1):91–106, Jan 2001.

[12] G. S. Chirikjian. Kinematics of a metamorphic robotic system. In *Robotics and Automation, Proceedings., 1994 IEEE International Conference on*, volume 1, pages 449–455.

[13] G. S. Chirikjian. Bounds for self-reconfiguration of metamorphic robots. In *Robotics and Automation, Proceedings, IEEE Int. Conf. on (ICRA)*, volume 2, pages 1452–1457, 1996.

[14] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, June 2005.

[15] D. J. Christensen and K. Støy. Selecting a meta-module to shape-change the atron self-reconfigurable robot. In *Robotics and Automation, Proceedings, IEEE Int. Conf. on (ICRA)*, pages 2532–2538, 2006.

[16] F. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[17] M. De Rosa, S. C. Goldstein, P. Lee, J. D. Campbell, and P. Pillai. Scalable shape sculpting via hole motion: Motion planning in lattice-constrained module robots. In *Robotics and Automation, Proceedings, IEEE Int. Conf. on (ICRA)*, May 2006.

[18] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Heidelberg, third edition, 2005.

[19] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1), 1989.

[20] R. Fitch and Z. Butler. Million module march: Scalable locomotion for large self-reconfiguring robots. *Int. J. Robobitc Research*, 27:331–343, March 2008.

[21] R. Fitch, Z. Butler, and D. Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. *Intelligent Robots and Systems*, 3:2460– 2467, 2003.

[22] R. Fitch, Z. Butler, and D. Rus. Reconfiguration planning among obstacles for heterogeneous self-reconfiguring robots. *Robotics and Automation*, pages 117 – 124, 2005.

[23] S. Fukuda, Toshio & Nakagawa. *Dynamically reconfigurable robotic system*, volume 1, pages 1581–1586. IEEE, 1988.

[24] T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss. Self organizing robots based on cell structures - CEBOT. In *Intelligent Robots, Proceeding, IEEE Int. Workshop on*, 1988.

[25] R. Ghrist. Shape complexes for metamorphic robot systems. *Proc. Workshop in Algorithmic Foundations of Robotics*, 2002.

[26] R. Ghrist and V. Peterson. The geometry and topology of reconfiguration. *Advances in Applied Mathematics*, 38:302–323, 2007.

[27] S. C. Goldstein, J. D. Campbell, and T. C. Mowry. Programmable matter. *IEEE Computer*, 38(6):99–101, June 2005.

[28] S. W. Golomb. Tiling with sets of polyominoes. *Journal of Combinatorial Theory*, 9(1):60 – 71, 1970.

[29] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.

[30] F. Hou and W.-M. Shen. On the complexity of optimal reconfiguration planning for modular reconfigurable robots. In *Robotics and Automation, Proceedings, IEEE Int. Conf. on (ICRA)*, 2010.

[31] M. Jorgensen, E. Ostergaard, and H. Lund. Modular ATRON: modules for a self-reconfigurable robot. In *Intelligent Robots and Systems, Proceeding, IEEE Int. Conf. on (IROS)*, 2004.

[32] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, Proceedings, IEEE Int. Conf. on (ICRA)*, pages 566–580, 1997.

[33] A. Kawakami, A. Torii, K. Motomura, and S. Hirose. Smc rover: Planetary rover with transformable wheels. In *Experimental Robotics, Int. Symposium on, (ISER)*, pages 498–506, 2002.

[34] B. Khoshnevis, P. Will, and W.-M. Shen. Highly compliant and self-tightening docking modules for precise and fast connection of self-reconfigurable robots. In *Robotics and Automation, Proceedings, IEEE Int. Conf. on (ICRA)*, pages 2311–2316, Taiwan, 2003.

[35] B. Kirby, B. Aksak, J. Campbell, J. Hoburg, T. Mowry, P. Pillai, and S. Goldstein. A modular robotic system using magnetic force effectors. *Intelligent Robots and Systems, Proceeding, IEEE Int. Conf. on (IROS)*, pages 2787–2793, 29 2007-Nov. 2 2007.

[36] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–679, 1983.

[37] J. A. G. Knight. The latest developments of FMS in Japan. *Flexible Manufacturing Systems, Proceedings of, Int. Conf. on,*, pages 31–47, 1982.

[38] J. Kuffner and S. LaValle. RRT-Connect: An efficient approach to single-query path planning. *Robotics and Automation, Proceedings, IEEE Int. Conf. on (ICRA)*, pages 995–1001, 2000.

[39] H. Kurokawa, A. Kamimura, E. Yoshida, K. Tomita, S. Kokaji, and S. Murata. M-TRAN II: Metamorphosis from a four-legged walker to a caterpillar. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2454–2459, 2003.

[40] H. Kurokawa, S. Murata, E. Yoshida, K. Tomita, and S. Kokaji. A three-dimensional self-reconfigurable system. *Advanced Robotics*, 13(6):591–602, 1998.

[41] H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, T. Hasuo, and S. Murata. Distributed self-reconfiguration of M-TRAN III modular robotic system. *Int. J. Robotics Research*, 27:373–386, 2008.

[42] H. Kurokawa, E. Yoshida, K. Tomita, A. Kamimura, S. Murata, and S. Kokaji. Deformable multi M-TRAN structure works as walker generator. In *Intelligent Autonomous Systems, Proceedings, IEEE Int. Conf. on (IAS)*, pages 746–753, 2004.

[43] T. Larkworthy and G. Hayes. Utilizing redundancy in modular robots to achieve greater accuracy. *Robot Communication and Coordination, Proceedings of, Int. Conf. on (Robocomm)*, 2009.

[44] T. Larkworthy, G. Hayes, and S. Ramamoorthy. General motion planning methods for self-reconfiguration planning. *Towards Autonomous Robotic Systems, Proceedings of, (TAROS)*, 2009.

[45] T. Larkworthy and S. Ramamoorthy. An effecient centralized algorithm for self-reconfiguration planning in a modular robot. In *Robotics and Automation, Proceedings, IEEE Int. Conf. on (ICRA)*, 2010.

[46] T. Larkworthy and S. Ramamoorthy. A characterization of the reconfiguration space of self-reconfiguring robotic systems. *Robotica*, 29(1):73–85, 2011.

[47] S. M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. *Cambridge University Press*, 1998.

[48] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

[49] H. Lipson. Principles of modularity, regularity, and hierarchy for scalable systems. In *In GECCO Workshop on Modularity, Regularity, and Hierarchy in Evolutionary Computation*, 2004.

[50] D. Martins and R. Simoni. Enumeration of planar metamorphic robots. *Reconfigurable Mechanisms and Robots, ASME/IFToMM Int. Conf. on, ReMAR*, pages 610–611, 2009.

[51] J. McLurkin and D. Yamins. Dynamic task assignment in robot swarms. *Proceedings of Robotics: Science and Systems, June*, 8, 2005.

[52] F. Mondada, G. C. Pettinaro, I. Kwee, L. Gambardella, D. Floreano, S. Nolfi, J. louis Deneubourg, and M. Dorigo. M.: Swarm-bot: A swarm of autonomous mobile robots with self-assembling capabilities. In *Self-Organisation and Evolution of Social Behaviour, Int. Workshop on,*, 2002.

[53] S. Murata and H. Kurokawa. Self-reconfigurable robots, shape-changing cellular robots can exceed conventional robot flexibility. *IEEE Robotics Automation Magazine*, 14:71–78, 2007.

[54] S. Murata, H. Kurokawa, and S. Kokaji. Self-assembling machine. In *Robotics and Automation, Proceedings, IEEE Int. Conf. on (ICRA)*, pages 441–448, 1994.

[55] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji. M-tran: self-reconfigurable modular robotic system. *Mechatronics, IEEE/ASME Transactions on*, 7(4):431–441, 2002.

[56] E. H. Ostergaard, K. Tomita, and H. Kurokawa. Distributed metamorphosis of regular M-TRAN structures. *Distributed Autonomous Robotic Systems*, 6:169–178, 2004.

[57] R. Oung, F. Bourgault, M. Donovan, and R. D'Andrea. The distributed flight array. pages 601–607, May 2010.

[58] A. Pamecha, I. Ebert-Uphoff, and G. Chirikjian. Useful metrics for modular robot motion planning. *Robotics and Automation, IEEE Transactions on*, 13(4):531–545, Aug 1997.

[59] K. Payne, J. Everist, F. Hou, and W.-M. Shen. Single-sensor probabilistic localization on the SeReS self-reconfigurable robot. In *Intelligent Autonomous Systems, Proceedings, IEEE Int. Conf. on (IAS)*, Tokyo, Japan, March 2006.

[60] K. Payne, B. Salemi, P. Will, and W.-M. Shen. Sensor-based distributed control for chain-typed self-reconfiguration. In *Intelligent Robots and Systems, Proceeding, IEEE Int. Conf. on (IROS)*, pages 2074–2080, Sendai, Japan, Sept./Oct. 2004.

[61] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, Mar. 1986.

[62] J. R. Quinlan. *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning)*. Morgan Kaufmann, 1 edition, Oct. 1992.

[63] R. J. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[64] M. Rubenstein, K. Payne, P. Will, and W.-M. Shen. Docking among independent and autonomous CONRO self-reconfigurable robots. In *Robotics and Automation, Proceedings, IEEE Int. Conf. on (ICRA)*, pages 2877–2882, New Orleans, USA, April/May 2004.

[65] D. Rus and M. Vona. Self-reconfiguration planning with compressible unit modules. *Robotics and Automation, Proceedings, IEEE Int. Conf. on (ICRA)*, 4:2513–2520, 1999.

[66] D. Rus and M. Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10(1):107–124, 2001.

[67] S. J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[68] B. Salemi, M. Moll, and W.-M. Shen. SUPERBOT: A deployable, multi-functional, and modular self-reconfigurable robotic system. In *Intelligent Robots and Systems, Proceeding, IEEE Int. Conf. on (IROS)*, Beijing, China, Oct. 2006. To appear.

[69] B. Salemi and W.-M. Shen. Distributed behavior collaboration for self-reconfigurable robots. In *Robotics and Automation, Proceedings, IEEE Int. Conf. on (ICRA)*, pages 4178–4183, New Orleans, USA, April/May 2004.

[70] B. Salemi, P. Will, and W.-M. Shen. Distributed task negotiation in self-reconfigurable robots. In *Intelligent Robots and Systems, Proceeding, IEEE Int. Conf. on (IROS)*, pages 2448–2453, 2003.

[71] W.-M. Shen and P. Will. Docking in self-reconfigurable robots. In *Intelligent Robots and Systems, Proceeding, IEEE Int. Conf. on (IROS)*, pages 1049–1054, 2001.

[72] K. Støy. Controlling self-reconfiguration using cellular automata and gradients. In *Intelligent Autonomous Systems, Proceedings, IEEE Int. Conf. on (IAS)*, pages 693–702, 2004.

[73] K. Støy and R. Nagpal. Self-reconfiguration using directed growth. In *Distributed Autonomous Robotic Systems, In Proceedings, Symp. on, (DARS)*, pages 1–10, 2004.

[74] J. W. Suh, S. B. Homans, and M. Yim. Telecubes: Mechanical design of a module for self-reconfigurable robotics. In *Robotics and Automation, Proceedings, IEEE Int. Conf. on (ICRA)*, volume 4, pages 4095–4101, 2002.

[75] S. Vassilvitskii, M. Yim, and J. W. Suh. A complete, local and parallel reconfiguration algorithm for cube style modular robots. In *Robotics and Automation, Proceedings, IEEE Int. Conf. on (ICRA)*, volume 1, pages 117–122, 2002.

[76] J. E. Walter, M. E. Brooks, D. F. Little, and N. M. Amato. Enveloping multi-pocket obstacles with hexagonal metamorphic robots. In *Robotics and Automation, Proceedings, IEEE Int. Conf. on (ICRA)*, pages 2204–2209, 2004.

[77] J. E. Walter, J. L. Welch, and N. M. Amato. Distributed reconfiguration of metamorphic robot chains. *Distrib. Comput.*, 17:171–189, August 2004.

[78] J. E. Walter, J. L. Welch, and N. M. Amato. Distributed reconfiguration of metamorphic robot chains. *Distributed Computing*, 17:171 – 189, 2004.

[79] P. J. White, K. Kopanski, and H. Lipson. Stochastic self-reconfigurable cellular robotics. In *Robotics and Automation, Proceedings, IEEE Int. Conf. on (ICRA)*, pages 2888–2893. IEEE Computer Society Press, 2004.

[80] G. D. Yim, M. and A. Casal. Connectivity planning for closed-chain reconfiguration. *SPIE, Sensor Fusion and Decentralized control in Robotic Systems*, 4196, Nov 2000.

[81] M. Yim. *Locomotion With A Unit-Modular Reconfigurable Robot*. PhD thesis, Dept. of Mech. Eng. Stanford Univ., 1994.

[82] M. Yim, W. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular self-reconfigurable robot systems – challenges and opportunities for the future. *IEEE Robotics and Automation Magazine*, March:43–53, 2007.

[83] E. Yoshida, S. Murata, A. Kamimura, K. Tomita, H. Kurokawa, and S. Kokaji. Evolutionary motion synthesis for a modular robot using genetic algorithm. *J. of Robotics and Mechatronics*, 15:227–237, 2003.

[84] E. Yoshida, S. Murata, A. Kamimura, K. Tomita, H. Kurokawa, and S. Kokaji. Evolutionary synthesis of dynamic motion and reconfiguration process for a modular robot. In *Robotics and Automation, Proceedings, IEEE Int. Conf. on (ICRA)*, pages 1004–1010, 2003.

[85] E. Yoshida, S. Murata, S. Kokaji, K. Tomita, and H. Kurokawa. Micro self-reconfigurable modular robot using shape memory alloy. *J. of Robotics and Mechatronics*, 13:212–219, 2001.